НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

«До захисту допущено»
В.О. завідувача кафедри
_____ О.Л. Тимощук

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 "Комп'ютерні науки"

на тему: «Наочне доведення теореми Хомського-Шютценберже в розширеному формулюванні»

Виконав:
студент IV курсу, групи КА-55
Мелентьєва Ада Денисівна

Керівник:
професор кафедри ММСА,
к.т.н. Спекторський І. Я.
_____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О. А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А.Є. _____
Рецензент:
к.т.н. Ільєнко А.Б _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

РЕФЕРАТ

Дипломна робота: 75 с., 6 табл., 29 рис., 2 дод., 20 джерел.

ТЕОРЕМА ХОМСЬКОГО-ШЮТЦЕНБЕРЖЕ, КОНТЕКСТНО-ВІЛЬНА МОВА, РЕГУЛЯРНА МОВА, МОВА ДІКА, ПРАВИЛЬНА ДУЖКОВА ПОСЛІДОВНІСТЬ, СКІНЧЕННИЙ АВТОМАТ, АВТОМАТ З МАГАЗИННОЮ ПАМ’ЯТТЮ, НОРМАЛЬНА ФОРМА ГРЕЙБАХ, ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ.

Темою даної роботи є: «Наочне доведення теореми Хомського-Шютценберже в розширеному формулюванні».

Метою даної роботи є доведення теореми Хомського-Шютценберже у розширеному формулюванні. У роботі використано такі елементи теорії формальних мов та автоматів, як скінченні автомати, автомати з магазинною пам’яттю, мова Діка.

Було розроблено програмний продукт, що проводить інтерактивне конструктивне доведення теореми. Він дозволяє перевірити справедливість теореми для будь-якої контекстно-вільної мови. Виконано тестування розробленого програмного продукту.

ABSTRACT

The thesis contains: 75 p., 6 tabl., 29 fig., 2 app., 20 sources.

CHOMSKY-SCHUTZENBERGER REPRESENTATION THEOREM, CONTEXT-FREE LANGUAGE, REGULAR LANGUAGE, DYCK LANGUAGE, BALANCED BRACKET SEQUENCE, FINITE STATE MACHINE, PUSHDOWN AUTOMATON, GREIBACH NORMAL FORM, FUNCTIONAL PROGRAMMING.

The theme of the thesis is as follows: «A Transparent Proof of the Extended Formulation of the Chomsky-Schutzenberger Representation Theorem».

The goal of the thesis is to prove the extended formulation of the Chomsky-Schutzenberger Representation Theorem. The work utilizes such concepts of the formal language and automata theory as finite state machines, pushdown automata, the Dyck language.

A software product that executes an interactive proof of the theorem has been developed. It allows to verify that the theorem holds for any given context-free language. Testing of the developed product has been performed.

ЗМІСТ

ВСТУП

Теорема Хомського-Шютценберже про представлення контекстно-вільних мов, хоч і не є загальновідомою, є тим не менш однією з найважливіших теорем про контекстно-вільні мови у тому сенсі, що вона дозволяє зробити неочевидні висновки щодо будови контекстно-вільних мов та пов’язати їх з іншими галузями теорії формальних мов, таких як регулярні мови (та, відповідно, регулярні вирази). Вперше її було сформульовано Н. Хомським та М. П. Шютценберже в [1]. Значення та застосування цієї теореми буде детально розглянуто у наступному розділі роботи.

На жаль, незважаючи на переваги вивчення цієї теореми, в усіх україномовних чи російськомовних джерелах, що містять її доведення, це доведення викладено складним та неочевидним чином, без пояснення основних ідей, що його зумовлюють. У багатьох таких джерелах (наприклад, [7]) розділи, що стосуються цієї теореми та суміжних понять, позначено як матеріал підвищеної складності. У англомовній літературі представлено більше різних варіантів доведення теореми, але переважна їх кількість все одно є складною для розуміння.

У 2012 році на конференції SAIT було представлено доведення цієї теореми за допомогою автоматів з магазинною пам’яттю [18]. Через формат конференції дане доведення було подано у стислому вигляді, тому воно теж є непростим для розуміння. До того ж, у даному доведенні було використано формулювання теореми більш вузьке, ніж у [7]. Втім, застосування автоматів з магазинною пам’яттю значно спрощує доведення та підкреслює основну його ідею. Тож метою цієї роботи є побудувати на основі [18] доведення розширеного формулювання теореми, детально пояснити його етапи та ідею, а також, оскільки доведення є конструктивним, створити ПЗ, яке дасть змогу продемонструвати алгоритм доведення на конкретних, заданих користувачем прикладах.

Можливим застосуванням результату роботи стане його використання у викладанні теорії формальних мов, адже більш просте доведення та ілюстративне ПЗ дадуть змогу краще пояснити деякі властивості контекстно-вільних мов у формі, доступній, наприклад, студентам молодших курсів ВНЗ.

Цінність доведення розширеного формулювання теореми за допомогою автоматів з магазинною пам’яттю полягає в тому, що воно підтверджує, що один з побудованих автоматів залежить виключно від розглядуваного алфавіту та може використовуватись у розпізнаванні слів будь-якої з контекстно-вільних мов над цим алфавітом. З точки зору програмного розпізнавання слів контекстно-вільних мов це означає менші витрати програмних ресурсів та прискорення швидкості розпізнавання слів декількох мов над одним і тим самим алфавітом. Хоч доведення цього формулювання вже було наведено у [7], його поєднання з ідеєю використання автоматів з магазинною пам’яттю дозволяє побудувати швидкий та нескладний у реалізації алгоритм представлення контекстно-вільної мови у формі, запропонованій теоремою, та розпізнавання її слів.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

Для того, щоб однозначно сформулювати та довести теорему Хомського-Шютценберже, потрібно спершу описати елементи теорії формальних мов та автоматів, що у ній згадуються.

1.1 Поняття, використані у формулюванні теореми Хомського-Шютценберже

Теорема Хомського-Шютценберже використовує певні базові поняття теорії формальних мов, без знання яких вона може виявитись складною для розуміння. У цьому підрозділі наведено короткі пояснення до головних з них. Джерелом пояснень послужили [15, 7, 17], хоча деякі позначення було замінено на більш сучасні.

1.1.1 Граматика, мова

Граматика G описується набором

$$G = (N, T, P, S), \tag{1.1}$$

де N — скінченний алфавіт нетермінальних символів;

T — скінченний алфавіт термінальних символів;

P — скінченна множина правил вигляду (α, β), де $\alpha \in N^+, \beta \in (N \cup T)^*$;

S — початковий символ, $S \in N$.

При цьому виконується

$$N \cap T = \varnothing \tag{1.2}$$

Правила з P також записуються як $\alpha \to \beta$ та позначають перехід від α до β.

Виведення $\alpha \overset{*}{\Rightarrow} \beta$ має місце, якщо існує послідовність правил, що дозволяє перейти від α до β.

Мова є набором слів, породженим граматикою:

$$L(G) = \{x \in T^* | S \overset{*}{\Rightarrow} x\} \tag{1.3}$$

1.1.2 Контекстно-вільна мова

Контекстно-вільна граматика вигляду (1.1) задовольняє такому обмеженню: кожне її правило має вигляд

$$A \to \beta, \tag{1.4}$$

де $A \in N$;

$\beta \in (N \cup T)^*$.

Контекстно-вільна мова — мова, що породжується контекстно-вільною граматикою.

1.1.3 Мова Діка

Мова Діка — спеціальна контекстно-вільна мова, що описує правильні дужкові послідовності n типів дужок. Її можна задати такою граматикою:

Мова Діка D над алфавітом

$$T = (a_1, b_1, a_2, b_2, \ldots a_n, b_n) \tag{1.5}$$

задається набором (N, T, P, S), де $N = \{S\}$, і P складається з правил:

$$S \to \varepsilon \tag{1.6}$$

$$S \to a_i S b_i S, i = \overline{1, n} \tag{1.7}$$

1.1.4 Скінченний автомат та регулярна мова

Детермінований скінченний автомат є абстрактною машиною, що отримує на вхід рядок символів, переходить між своїми внутрішніми станами залежно від цих символів, та кінець кінцем вирішує, прийняти чи відкинути цей рядок. Він описується п'ятіркою

$$M = (Q, T, \Delta, I, F), \tag{1.8}$$

де Q — множина станів;

T — алфавіт вхідних символів;

Δ — функція переходів між станами;

$I \subset Q$ — множина початкових станів (зазвичай, один);

$F \subset Q$ — множина кінцевих станів.

Функція переходів Δ є множиною наборів $((p, \alpha), (q))$, де $p, q \in Q, \alpha \in T$. Кожний з таких наборів позначає перехід $p \xrightarrow{\alpha} q$ зі стану p до стану q, якщо поточний розглядуваний символ — α.

Рядок s приймається автоматом A, якщо після застосування Δ до усіх його символів по порядку можливо опинитись у $q \in F$.

Недетермінований скінченний автомат відрізняється від детермінованого тим, що його функція переходів дозволяє з одного й того самого стану за наявності одного й того самого вхідного символа переходити у декілька різних станів, таким чином, один і той самий вхідний рядок може бути оброблено кількома різними способами. Рядок s приймається таким автоматом, якщо хоча б один з цих способів обробки приводить автомат у кінцевий стан.

Клас детермінованих та недетермінованих скінченних автоматів є еквівалентним: кожен детермінований скінченний автомат є частковим випадком недетермінованого, а для кожного недетермінованого скінченного автомата можна побудувати еквівалентний йому детермінований.

Регулярна мова, що відповідає автомату A, складається з множини рядків, які приймає цей автомат.

1.1.5 Перетин мов

Перетин мов L_1 та L_2, оскільки мови, фактично, є множинами слів, є просто множиною $L_1 \cap L_2$ з точки зору теорії множин. Тобто слово s належить $L_3 = L_1 \cap L_2$ тоді і тільки тоді, коли воно одночасно належить і мові L_1, і мові L_2 — еквівалентно, коли воно породжується обома граматиками, що задають ці мови.

1.2 Формулювання теореми Хомського-Шютценберже

Теорему, що розглядається у даній дипломній роботі, було вперше сформульовано Н. Хомським та М. П. Шютценберже у 1963 році [1]. Це одна з двох теорем, що носять ім'я Хомського-Шютценберже; її повна назва — теорема Хомського-Шютценберже про представлення контекстно-вільних мов. Інша з двох теорем стосується кількості слів заданої довжини, породжених контекстно-вільною мовою, та не має прямого зв'язку з розгладуваною в даній роботі теоремою [10, 13].

1.2.1 Оригінальне формулювання теореми

Оригінальне формулювання теореми [1] використовує визначення, що більше не зустрічатимуться у даній роботі, а деякі поняття з нього нині мають інші загальноприйняті назви, тому у даній роботі буде подано переформульовану, але еквівалентну, версію:

Теорема 1. Кожна контекстно-вільна мова L над скінченним алфавітом T однозначно задається: скінченним алфавітом T_1, гомоморфізмом

$$f : T_1^* \to T^*, \tag{1.9}$$

регулярною мовою $L_0 \subset T_1^*$, мовою Діка $D \subset T_1^*$, та правилом:

$$L = f(L_0 \cap D) \tag{1.10}$$

Тобто, будь-яка контекстно-вільна мова є, з точністю до певного гомоморфізму [2], перетином регулярної мови та мови Діка.

1.2.2 Розширене формулювання теореми

В [7] подано, з точністю до позначень, таке, більш широке, формулювання:

Теорема 2. Нехай дано алфавіт T. Тоді існують алфавіт T_1, контекстно-вільна мова $D \subseteq T_1$ та гомоморфізм h множини T_1^* на множину T^*, такі, що для будь-якої контекстно-вільної мови $L \subseteq T^*$ існує регулярна мова $L_0 \subseteq T_1^*$, для якої виконується

$$L = h(L_0 \cap D) \tag{1.11}$$

Зауважимо, що хоч у формулюванні не вказано, що D є мовою Діка, подане доведення використовує саме таку мову, а отже можемо переформулювати теорему так:

Теорема 3. Нехай дано алфавіт T. Тоді існують алфавіт T_1, мова Діка $D \subseteq T_1$ та гомоморфізм h множини T_1^* на множину T^*, такі, що для будь-якої контекстно-вільної мови $L \subseteq T^*$ існує регулярна мова $L_0 \subseteq T_1^*$, для якої виконується

$$L = h(L_0 \cap D) \tag{1.12}$$

Нескладно побачити, що формулювання 3 є узагальненням формулювання 1: так, можемо бачити, що алфавіт T_1, мова Діка D та гомоморфізм h

однозначно задаються вже початковим алфавітом T, в той час як від контекстно-вільної мови L залежить тільки регулярна мова L_0.

Коли у наступному розділі більш детально розглядатиметься зв'язок між контекстно-вільними мовами та автоматами з магазинною пам'яттю, таке розмежування алфавіта і мови зможе бути використано для кращого розуміння цього зв'язку.

1.3 Значення теореми Хомського-Шютценберже

Теорема Хомського-Шютценберже є важливою віхою у дослідженні контекстно-вільних мов, адже вона дає розуміння внутрішньої структури контекстно-вільних мов та показує, що ту особливість, що відрізняє контекстно-вільну мову від регулярної, можна розглядати окремо від конкретних правил цієї мови: її «контекстно-вільність» можна передати мовою Діка, що залежить лише від алфавіту контекстно-вільної мови, а конкретні правила можна після цього передати за допомогою регулярної мови.

Завдяки своїй відносній простоті теорема Хомського-Шютценберже може використовуватись під час викладання теорії формальних мов та автоматів студентам ВНЗ, щоб показати взаємозв'язок між частинами теорії та підвищити інтерес студентів до предмету.

Теорема Хомського-Шютценберже дозволяє знайти альтернативне доведення кільком іншим теоремам, пов'язаним з формальними мовами. Зокрема, вона може використовуватись для альтернативного доведення теореми Паріка [14, 8]. Теорема Паріка стверджує, що якщо у словах контекстно-вільної мови розглядати лише кількості символів без урахування їхнього порядку, то така мова є регулярною. Ідея доведення за допомогою теореми Хомського-Шютценберже полягає в тому, щоб довести, що мова $L_0 \cap D$ є напівлінійною.

Окрім цього, існує узагальнення теореми Хомського-Шютценберже на клас зважених контекстно-вільних автоматів на множинні контекстно-віль-

ні граматики (multiple context-free grammars), що розглядають нетермінальні символи граматик як предикати довільної арності [4]. Граматики цього класу можуть бути представлені як перетик множинної мови Діка та звичайної регулярної мови.

1.4 Постановка задачі дослідження

Задачею даного дослідження є отримати доведення теореми Хомського-Шютценберже у розширеному формулюванні з використанням автоматів з магазинною пам’яттю, а також створити програмний продукт, що повторюватиме конструктивний алгоритм доведення та виконуватиме описані у доведенні побудови для довільних алфавіту та контекстно-вільної мови з можливістю інтерактивної взаємодії.

1.5 Висновки

Теорема Хомського-Шютценберже є важливою теоремою теорії контекстно-вільних та взагалі формальних мов. Її можна використовувати при викладанні теорії формальних мов для пояснення зв’язку між контекстно-вільними та регулярними мовами. Ій присвячено доволі мало україномовної літератури, тож метою даної роботи є виправити цей недолік, а також створити інтерактивний матеріал, що стосується цієї теореми, у вигляді програмного продукта.

2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

Метою даного розділу є розглянути доведення теореми Хомського-Шютценберже, що використовує автомати з магазинною пам'яттю (автомати зі стеком). Таке доведення вже було представлено у [18], але у вузькому формулюванні (формулюванні 1) та стисло. Тож цей розділ покликаний розглянути взаємозв'язок контекстно-вільних мов та автоматів з магазинною пам'яттю та доречність використання останніх для доведення теореми Хомського-Шютценберже, а також розширити доведення до формулювання 3, використавши ідеї доведення Гінзбурга [7].

2.1 Зв'язок контекстно-вільних мов та автоматів з магазинною пам'яттю

Зв'язок контекстно-вільних мов та автоматів з магазинною пам'яттю є загальновідомим, але його варто згадати, оскільки він є центральним для розглядуваного доведення теореми Хомського-Шютценберже та, відповідно, алгоритму побудови відповідних регулярної мови та мови Діка, а також автоматів з магазинною пам'яттю, що їм відповідають.

2.1.1 Нормальна форма Грейбах контекстно-вільної граматики

Нормальна форма Грейбах — форма запису контекстно-вільної граматии, у якій кожне правило має вигляд

$$A \to aA_1A_2 \dots A_n, \tag{2.1}$$

де $A, A_1, A_2 \dots A_n$ — нетермінальні символи;

a — термінальний символ.

Єдиним можливим виключенням є правило

$$S \to \varepsilon, \tag{2.2}$$

де S — стартовий нетермінальний символ;

ε — пустий символ.

Це правило дозволяє мові включати в себе пустий рядок.

Кожну контекстно-вільну граматику можна записати у нормальній формі Грейбах так, що мова, яку вона описує, не зміниться [9]. Детальний алгоритм наведено у розділі, присвяченому алгоритмові розв'язку задачі роботи. Цей алгоритм також можна знайти в [12].

2.1.2 Автомат з магазинною пам'яттю

Автомат з магазинною пам'яттю є узагальненням скінченного автомата [16]. Окрім станів, він також має стек (магазинну пам'ять), куди під час переходів можуть записуватися символи окремого алфавіту; верхній з цих символів використовується як один з аргументів для функції переходу, причому при переході він прибирається.

Автомат з магазинною пам'яттю описується сімкою

$$M = (Q, T, \Gamma, \Delta, I, F, S) \tag{2.3}$$

де Q — множина станів;

T — алфавіт вхідних символів;

Γ — алфавіт символів стеку;

Δ — функція переходів між станами;

$I \subset Q$ — множина початкових станів (зазвичай, один);

$F \subset Q$ — множина кінцевих станів;

S — стартовий символ стеку.

Функція переходів Δ є множиною наборів $((p, \alpha, \beta), (q, \gamma))$, де $p, q \in Q, \alpha \in T, \beta \in \Gamma, \gamma \in \Gamma^*$. Кожний з таких наборів позначає перехід $p \xrightarrow{\alpha} q$ зі стану p до стану q, за умови, що поточний розглядуваний символ — α, а верхній символ стеку — β. При цьому β прибирається зі стеку, і замість нього у стек

додається один або декілька символів — γ. Якщо замість γ стоїть ε, це означає, що у стек не додається символ.

Існує два варіанти визначення, що рядок s приймається автоматом A:

а) Якщо після застосування Δ до усіх його символів по порядку можливо опинитись у $q \in F$.

б) Якщо після застосування Δ до усіх його символів по порядку можливо досягти ситуації, коли стек автомата пустий.

Ці визначення є еквівалентними в тому сенсі, що для кожного автомата, що використовує визначення 1, існує автомат, що використовує визначення 2 і приймає таку саму множину рядків, і навпаки. У даній роботі розглядатимуться автомати, що використовують визначення 2. Відповідно, визначати множину кінцевих станів не буде обов'язковим.

2.1.3 Зв'язок автоматів з магазинною пам'яттю та контекстно-вільних мов

Для кожної контекстно-вільної граматики можна побудувати автомат з магазинною пам'яттю, що розпізнає мову, яка задається цією граматикою. Для граматики, записаної у нормальній формі Грейбах, алгоритм є простим.

Нехай G — контекстно-вільна граматика у нормальній формі Грейбах, тобто її множина правил P складається з правил вигляду (2.1), а також, можливо, правила (2.2).

Побудуємо автомат з магазинною пам'яттю, у якого $Q = \{q_0\}, I = q_0, \Gamma = N$ (з визначення граматики), а N та S співпадають з такими з визначення граматики. Таким чином, стек автомата міститиме нетермінальні символи граматики, причому він ініціалізується її стартовим символом; тим часом алфавіт рядків, які автомат розглядатиме, співпадає з алфавітом рядків, які породжує граматика.

Залишається описати правила переходу Δ. Щоб отримати автомат, що розпізнає ті й тільки ті рядки, що породжуються граматикою G, задамо Δ так, щоб мала місце така властивість: якщо у процесі виконання переходів з G можна отримати проміжний стан $a_1a_2\dots a_nA_1A_2\dots A_m$ $(a_i \in T, A_i \in N)$, то після розпізнавання автоматом рядка $a_1a_2\dots a_n$ можна дійти до стану, коли стек складається, згори донизу, з символів $A_1A_2\dots A_m$ — і навпаки. Таким чином, автомат завжди «розгортатиме» нетермінальні символи зліва направо, оскільки він читає стек згори донизу.

Оскільки граматику записано у нормальній формі Грейбах, при «розгортанні» нетермінальних символів зліва направо завжди отримуватимемо стан вигляду $a_1a_2\dots a_nA_1A_2\dots A_m$. Оскільки контекстно-вільні граматики не є чутливими до порядку застосування правил, маємо взаємно однозначну відповідність між проміжними станами породження слова контекстно-вільної мови та станами автомата з магазинною пам'яттю. При цьому автомат прийматиме рядок тоді і тільки тоді, коли його стек є порожнім, тобто для $a_1a_2\dots a_nA_1A_2\dots A_m$ маємо $m = 0$, тобто слово повністю складається з термінальних символів, а отже, є словом контекстно-вільної мови, породженої граматикою G.

Щоб задовольнити дану властивість, достатньо кожному правилу вигляду (2.1) з G поставити у відповідність перехід $((p, \alpha, \beta), (q, \gamma)) \in \Delta$, де $\alpha = a, \beta = A, \gamma = (A_1, A_2, \dots A_n), p = q = q_0$, таким чином, що A_1 тепер стане верхнім символом стека.

Якщо граматика містить правило (2.2), додамо також правило $((q_0, \varepsilon, S)(q_0, \varepsilon))$. Очевидно, що за допомогою цього правила автомат зможе приймати пусті рядки. Зауважимо, що це єдиний ε-перехід у правилах автомата, і він використовується виключно для прийняття пустого рядка, тобто щонайбільше один раз (за побудовою нормальної форми Грейбах, стартовий символ S зустрічається лише у лівій частині правил). Отже, у алгоритмі

розпізнавання слова за допомогою даного автомата з магазинною пам'яттю не може виникнути нескінченний цикл.

Таким чином, маємо алгоритм побудови автомата з магазинною пам'яттю, еквівалентного заданій контекстно-вільній граматиці.

2.2 Порівняння існуючих доведеннь теореми Хомського-Шютценберже без допомоги та за допомогою автоматів з магазинною пам'яттю

У даній роботі розглядатимуться два існуючих доведення теореми Хомського-Шютценберже: доведення С. Гінзбургом розширеного формулювання теореми без допомоги автоматів з магазинною пам'яттю [7] та доведення Поліщуком та ін. оригінального формулювання теореми за допомогою автоматів з магазинною пам'яттю [18]. Спершу наведемо основні ідеї обох доведень, а потім порівняємо їх та зробимо висновки щодо того, яким чином, спираючись на них, краще за все побудувати доведення розширеного формулювання за допомогою автоматів з магазинною пам'яттю.

2.2.1 Основна проблема теореми Хомського-Шютценберже

Теорема Хомського-Шютценберже представляє контекстно-вільну мову як перетин регулярної мови та мови Діка (що є контекстно-вільною), причому у широкому формулюванні мова Діка не залежить від цієї контекстно-вільної мови, а лише від її алфавіту. Очевидно, будь-яку контекстно-вільну мову неможливо, у широкому формулюванні, представити лише регулярною мовою або лише мовою Діка: клас регулярних мов, як відомо, вужчий за клас контекстно-вільних мов (наприклад, $\{0^n 1^n, n \geqslant 0\}$ є контекстно-вільною, але не регулярною мовою); мова Діка, за широким формулюванням теореми, залежить лише від алфавіта, тож не може містити в собі інформацію про структуру будь-якої конкретної контекстно-вільної мови з цим алфавітом.

Отже, є необхідність якось «розподілити» інформацію, що задає контекстно-вільну мову, між регулярною мовою та мовою Діка, при цьому мова Діка має містити інформацію, що притаманна усім контекстно-вільним мовам з даним алфавітом, а регулярна мова — інформацію, що не виходить за межі класу регулярних мов. Слід також зауважити, що певна додаткова інформація може міститись у власне словах обох цих мов (адже гомоморфізм, що перетворюватиме ці рядки на рядки початкового алфавіту, буде одностороннім).

2.2.2 Порівняння різниці між регулярними та контекстно-вільними мовами з різницею між скінченними автоматами та автоматами з магазинною пам'яттю

Як згадувалось вище, регулярні мови є підмножиною контекстно-вільних мов, а скінченні автомати, в певному сенсі, — виродженим випадком автоматів з магазинною пам'яттю. При цьому кожній регулярній мові можна поставити у відповідність скінченний автомат, а кожній контекстно-вільній — автомат з магазинною пам'яттю. Отже, різниця між класами регулярних та контекстно-вільних мов має певним чином відображатись у різниці між скінченними автоматами та автоматами з магазинною пам'яттю.

Поглянемо на ситуацію з такого боку: клас регулярних мов співпадає з класом мов, породжених праволінійними граматиками [7]. Отже, кожну регулярну граматику можна звести до правил вигляду

$$A \to xB, \tag{2.4}$$

де $A, B \in N$;

$x \in T$, за виключенням, можливо. правила (2.2).

Як було розглянуто вище, кожну контекстно-вільну граматику можна звести до нормальної форми Грейбах, у якій кожне правило має вигляд (2.1) (за виключенням, можливо, правила (2.2)). Оскільки будь-яка граматика у

нормальній формі Грейбах є контекстно-вільною, клас контекстно-вільних граматик співпадає з класом граматик у нормальній формі Грейбах.

З вищенаведеного випливає, що різниця між класами регулярних та контекстно-вільних граматик еквівалентна різниці між класами праволінійних граматик та граматик у нормальній формі Грейбах. Як нескладно помітити, ця різниця полягає в можливій кількості нетермінальних символів у правій частині правил граматики: для праволінійної граматики такий символ може бути лише один, а для граматики у нормальній формі Грейбах їх може бути довільна скінченна кількість.

Що ж до різниці між скінченними автоматами та автоматами з магазинною пам'яттю, то вона, очевидно, полягає в наявності магазинної пам'яті (стеку). Зауважимо, що можливий альтернативний спосіб прийняття рядка автоматом з магазинною пам'яттю (у випадку, коли стек пустий) є еквівалентним прийняттю рядка у випадку, коли автомат знаходиться у кінцевому стані, а тож це можна не вважати різницею між даними класами автоматів.

Логічним видається провести паралель між наявністю стека та можливістю використовувати правила з кількома нетермінальними символами. Оскільки для контекстно-вільної граматики порядок застосування правил не є важливим, без обмеження загальності застосовуватимемо їх зліва направо (тобто кожного разу для найлівішого нетермінального символа). Тоді на будь-якому кроці матимемо слово вигляду $a_1 a_2 \dots a_n A_1 A_2 \dots A_m$. У регулярної граматики при цьому на кожному етапі творення слова буде, за рахунок праволінійності, максимум один нетермінальний символ, тоді як у контекстно-вільної таких символів може бути декілька. Оскільки кожне застосування правила відбувається, за нашою домовленістю, до найлівішого нетермінального символа, та породжує один термінальний символ та один чи кілька нетермінальних (дописуючи їх зліва), видно, що список $A_1 A_2 \dots A_m$ можна в певному сенсі розглядати як стек: при кожному переході ми розглядаємо лише перший з нетермінальних символів (верхній елемент стеку), прибираємо його, а потім додаємо один або декілька

(можливо, жодного) нетермінальних символів зліва — елементів на початок стека. Такі самі міркування було використано вище під час опису побудови автомата з магазинною пам'яттю для заданої контекстно-вільної граматики.

Як було вказано вище, отриманий автомат з магазинною пам'яттю при цьому має один стан, тобто фактично не використовує механізм станів для того, щоб розглядати вхідні рядки.

Тим часом для регулярних мов, якщо провести аналогію з контекстно-вільними, у «стеку» завжди має бути лише один елемент (або жодного). Оскільки стека у скінченних автоматів немає, цей елемент потрібно пам'ятати якимось іншим чином. Тут якраз і можна використати функціонал станів, а саме створити по стану на кожний нетермінальний символ та знаходитись у цьому стані тоді, коли у проміжному слові граматики наявний цей нетермінальний символ. Відповідно, кінцевий стан позначатиме відсутність у слові нетермінальних символів — кінець його виведення.

Повертаючись до проблеми теореми Хомського-Шютценберже, її можна тепер описати так: необхідно за допомогою мови Діка та регулярної мови знайти спосіб зберігати елементи «стеку» нетермінальних символів у проміжному слові, причому мова Діка не може залежати від цих нетермінальних символів.

2.2.3 Порівняння мови Діка та операцій зі стеком

Якщо розглянути стек та можливі операції з ним, можна помітити, що декілька таких операцій, що починаються і завершуються порожнім стеком, можуть бути записані у вигляді правильної дужкової послідовності. Справді, якщо для кожного можливого елемента стека a задати пару дужок $[_a,]_a$, то відкриваюча дужка може використовуватись для позначення операції додавання цього елемента до стеку, а закриваюча — вилучення його зі стеку. При цьому правилу «зі стека можна вилучити лише той символ, який туди було додано останнім» відповідає правило «під час запису дужкової послідовності

зліва направо закриваючу дужку можна поставити лише парну до останньої відкритої». Таким чином, для заданого алфавіта T пар символів мова Діка над ним задає усі можливі сценарії взаємодії над стеком, що може містити елементи, кожний з яких відповідає парі символів алфавіта T.

2.2.4 Ідея доведення С. Гінзбурга

У доведенні розширеного формулювання теореми без використання автоматів з магазинною пам'яттю [7] контекстно-вільна граматика розглядається не у нормальній формі Грейбах, а у нормальній формі Хомського (правила мають вигляд $X \to a, A \to BC$ або $S \to \varepsilon$) [3].

Ідея доведення така: для кожного з правил $A \to BC$ контекстно-вільної мови додавати до регулярної мови правило вигляду $A \to X_{C[}B$, де $X_{C[}$ — спеціальна послідовність термінальних символів, що (фактично) символізує додавання символу C до стеку. Обрана послідовність виглядає таким чином: $dc^i d$, де d, c — спеціальні символи, додані до алфавіту T_1 разом зі своїми парами d', c'; i, фактично, нумерує нетермінальний символ.

Після цього, для кожного правила $X \to a$ до регулярної мови додається власне це правило (з символом, парним до термінального, аби не порушувати дужкову послідовність: $X \to aa'$), а також усі можливі правила вигляду $X \to aX_{C]}C$ для всіх C з правил $A \to BC$. Тут $X_{C]}$ знову складається лише з термінальних символів та має вигляд $d'c'^i d'$ (i — знову номер нетермінального символа). Таким чином, після виведення термінального символа (моменту, коли у контекстно-вільній граматиці при виведенні зліва направо варто було б перейти до наступного нетермінального символа) у цієї регулярної граматики є можливість «узяти» зі стеку будь-який нетермінальний символ та перейти до його розглядання.

Можна бачити, що отримана регулярна мова після гомоморфізму, що прибере усі пари символів, а також допоміжні символи c, d, буде надмножиною

початкової контекстно-вільної мови: оскільки регулярна граматика не дає змоги «запам'ятати» ще не розглянуті нетермінальні символи, коректний механізм стека не буде реалізовано. А саме, оскільки описані правила дають змогу після термінального символа додати будь-який нетермінальний, за аналогією це означатиме, що незалежно від поточного стану стека, з нього завжди можна буде взяти будь-який можливий символ. Відповідно, така граматика породить усі слова, що під час побудови «коректно використовували стек» (тобто початкову контекстно-вільну мову), а також усі слова, що «некоректно використовували стек» та не входять до контекстно-вільної мови.

Саме для того, щоб відсіяти усі слова, що «некоректно використовують стек», і використовується перетин з мовою Діка. Нескладно помітити, що послідовності $X_{C[}$ та $X_{C]}$ для кожного C можуть, фактично, вважатись просто ще одним типом дужок: для будь-яких A, B, що є правильними дужковими послідовностями, послідовність $X_{C[}AX_{C]}B$ теж буде правильною. Таким чином, слово, породжене розглянутою регулярною мовою, буде правильною дужковою послідовністю (а відповідно, входити до мови Діка) тоді й лише тоді, коли «використання» стеку буде коректним, а саме правило вигляду $X \to aX_{C]}C$ буде використовуватись лише тоді, коли останній символ, «запам'ятований» за допомогою правила $A \to X_{C[}B$ та ще не розглянутий, буде тим самим C. Це означатиме, що використання кожного правила вигляду $X \to BC$ початкової контекстно-вільної мови буде «змодельовано» коректно, а отже рівність контекстно-вільної мови та перетину регулярної мови з мовою Діка доведено.

Звісно, у джерелі [7] доведення проведено більш строго, але основна його ідея саме така, як подано вище.

2.2.5 Ідея доведення Поліщука та ін.

У доведенні оригінального формулювання теореми з використанням автоматів з магазинною пам'яттю [18] усі побудовані в ході доведення компонен-

ти залежать не тільки від початкового алфавіту T, а й від вихідної контекстно-вільної мови L. Немає сенсу детально розглядати в даній роботі їхню побудову, оскільки за використання розширеного формулювання теореми цей підхід не можна застосувати. Тим не менш, важливою у цьому доведенні є ідея представити усі три мови автоматами. Відповідно, використовуються автомати з магазинною пам'яттю для контекстно-вільних мов L' (мови, що використовує алфавіт T_1 та для якої $f(L') = L$) та D, і скінченний автомат для регулярної мови L_0. Далі проводиться доведення, у якому індуктивно показується, як саме кожен стан автомата для мови L' відповідає станам автоматів для мов L_0 та D.

Для опису стану автомата з магазинною пам'яттю використовується трійка $\langle q, s, \gamma \rangle$, де q — поточний стан автомата, s — рядок, який на даний момент залишилося розглянути, γ — поточний вміст стека. Для опису стана скінченного автомата використовується пара $\langle q, s \rangle$, оскільки стека у нього немає.

Твердження еквівалентності виглядає так:

$$\langle q_0, wv, \varepsilon \rangle \underset{M_1}{\vdash^*} \langle q, v, \gamma \rangle \iff (\langle q_0, wv \rangle \underset{M_2}{\vdash^*} \langle q, v \rangle) \wedge (\langle q_0, wv, \varepsilon \rangle \underset{M_3}{\vdash^*} \langle q_0, v, \gamma \rangle) \quad (2.5)$$

Фактично, воно показує еквівалентність множини станів автомата M_1 (що описує L') та перетину множин станів автоматів M_2 та M_3 (що описують L_0 та D, відповідно) після обробки префікса w рядка wv.

Дане твердження доводиться індуктивно з індукцією по довжині префікса w, а його очевидним наслідком є еквівалентність мови, що розпізнається автоматом M_1, та перетину мов, що розпізнаються автоматами M_2 і M_3. У свою чергу, наслідком цього твердження є власне теорема Хомського-Шютценберже.

2.3 Доведення розширеного формулювання теореми Хомського-Шютценберже за допомогою автоматів з магазинною пам'яттю

Описавши існуючі ідеї доведення теореми Хомського-Шютценберже, можна тепер розглянути, яким чином їх можна скомбінувати для того, щоб отримати доведення теореми Хомського-Шютценберже у розширеному формулюванні за допомогою автоматів з магазинною пам'яттю.

2.3.1 Ідея доведення

Проведемо доведення таким чином: візьмемо з доведення С. Гінзбурга ідею описувати «стек» послідовностями термінальних символів, але адаптуємо її для контекстно-вільних граматик у нормальній формі Грейбах, а не Хомського. Це дозволить нам у явному вигляді побудувати автомати для усіх трьох мов (до речі, автомат для мови Діка ми зможемо побудувати, ще знаючи тільки алфавіт T). Після цього ми сформулюємо та індуктивно доведемо твердження, аналогічне твердженню еквівалентності з доведення Поліщука та ін. (взаємозв'язок між станами автоматів вийде трохи інший), з якого безпосередньо випливатиме теорема Хомського-Шютценберже.

2.3.2 Розглядання кількох вхідних символів підряд під час переходу

Для спрощення доведення узагальнимо розглядувані правила переходу для автоматів так, щоб вони могли зчитувати кілька вхідних символів підряд для одного переходу.

Для скінченних автоматів це означатиме, що іхні правила переходу тепер матимуть вигляд $((p, \alpha), (q))$, де $p, q \in Q, \alpha \in T^*$.

Покажемо еквівалентність такого формулювання звичайному: для кожного правила $((p, \alpha_1\alpha_2 \dots \alpha_n), (q))$, де $\alpha_i \in T$, мжона побудувати $n-1$ допоміжний стан $q_1, q_2, \dots q_n$, що не буде ні початковим, ні кінцевим, а також такі правила

переходу:

$$\begin{cases} p \xrightarrow{\alpha_1} q_1 \\ q_i \xrightarrow{\alpha_{i+1}} q_{i+1}, i \in [1, n-1] \\ q_n \xrightarrow{\alpha_n} q \end{cases} \tag{2.6}$$

Для автоматів з магазинною пам'яттю це означатиме, що іхні правила переходу тепер матимуть вигляд $((p, \alpha, \beta), (q, \gamma))$, де $p, q \in Q, \alpha \in T^*, \beta \in \Gamma, \gamma \in \Gamma^*$.

Покажемо еквівалентність такого формулювання звичайному: для кожного правила $((p, \alpha_1\alpha_2 \dots \alpha_n, \beta), (q, \gamma))$, де $\alpha_i \in T$, можна побудувати $n-1$ допоміжний стан $q_1, q_2, \dots q_n$, що не буде ні початковим, ні кінцевим, а також такі правила переходу:

$$\begin{cases} (p, \alpha_1, \beta) \to (q_1, \beta) \\ (q_i, \alpha_{i+1}, \beta) \to (q_{i+1}, \beta), i \in [1, n-1] \\ (q_n, \alpha_n, \beta) \to (q, \gamma) \end{cases} \tag{2.7}$$

Легко бачити, що такі два підходи є еквівалентними.

Наслідком цього узагальнення є можливість легко будувати автомат з магазинною пам'яттю для мови, чиї правила мають вигляд

$$A \to a_1 a_2 \dots a_n A_1 A_2 \dots A_m, \tag{2.8}$$

тобто мову представлено у нормальній формі Грейбах за винятком того, що на початку правих частин правил можуть стояти декілька термінальних символів.

Зауважимо, що описане узагальнення не додає до автомата ε-переходів.

2.3.3 Мова T_1 та гомоморфізм f

Почнемо доведення теореми. Маючи алфавіт T, задамо алфавіт T_1 та гомоморфізм

$$f : T \to T_1 \tag{2.9}$$

Як у доведенні С. Гінзбурга, для кожного символа з T включимо його самого та парний до нього символ, а також дві пари додаткових символів, для яких оберемо $[,]$ та $(,)$. Отже:

$$T_1 = \{a, a' | a \in T\} \cap \{[,], (,)\} \tag{2.10}$$

Гомоморфізм f прибиратиме символи, яких немає у алфавіті T. Для $x \in T_1$:

$$f(x) = \begin{cases} x, x \in T \\ \varepsilon, x \notin T \end{cases} \tag{2.11}$$

2.3.4 Мова Діка D та автомат з магазинною пам'яттю M_D, що їй відповідає

Граматика мови Діка має розпізнавати усі правильні дужкові послідовності з пар символів алфавіту T_1. За поданим вище визначенням, її правила повинні мати вигляд або (2.2), або:

$$S \to aSa'S, \tag{2.12}$$

де a, a' — пара символів алфавіту T_1.

Але для того, щоб побудувати для цієї мови автомат з магазинною пам'яттю, було б зручно звести ці правила до нормальної форми Грейбах. Розглянемо зараз конкретну пару a, a' символів алфавіту T_1 та правило вигляду (2.12), що їй відповідає.

Введемо нетермінальний символ A та замінимо правило (2.12) такими двома правилами:

$$S \to aSA, \tag{2.13}$$

$$A \to a'S \tag{2.14}$$

Очевидно, заміна є еквівалентною.

Щоб граматика була у нормальній формі Грейбах, потрібно також впевнитись, що стартовий символ не зустрічається у правих частинах правил, та що єдине правило, що містить у правій частині ε — це правило (2.2).

Для цього спершу замінимо стартовий символ на нетермінальний символ S_1, додавши правило

$$S_1 \to S \tag{2.15}$$

Маємо правила:

$$S_1 \to \varepsilon, \tag{2.16a}$$

$$S_1 \to S, \tag{2.16b}$$

$$S \to \varepsilon, \tag{2.16c}$$

$$S \to aSA, \tag{2.16d}$$

$$A \to a'S \tag{2.16e}$$

Тепер приберемо правило (2.16c). Для цього необхідно до кожного правила з S у правій частині додати аналогічне правило, де S замінено пустим рядком. Матимемо:

$$S_1 \to \varepsilon, \tag{2.17a}$$

$$S_1 \to S, \tag{2.17b}$$

$$S \to aSA|aA, \tag{2.17c}$$

$$A \to a'S|a' \tag{2.17d}$$

Залишається прибрати перехід (2.17b). Це можна зробити, додавши усі правила з S у правій частині до правил для S_1:

$$S_1 \to aSA|aA|\varepsilon, \tag{2.18a}$$

$$S \to aSA|aA, \tag{2.18b}$$

$$A \to a'S|a' \tag{2.18c}$$

Таким чином, правилами для граматики мови Діка стануть наведені вище правила для усіх пар a, a' з мови T_1.

Тепер побудуємо автомат з магазинною пам'яттю M_D. За описаним вище алгоритмом це буде автомат $(Q, T_1, \Gamma, \Delta, I, S_1)$, де $Q = \{q_0\}$, $I = q_0$, $\Gamma = \{S_1, S\} \cup \{A_i | a_i, a_i' \in T_1\}$. Тепер опишемо правила Δ. Оскільки стан q_0 єдиний, а тому буде початковим і кінцевим станом кожного правила, можемо просто вказати для кожного правила вхідний символ та стани верхівки стеку:

$$S_1 \xrightarrow{\varepsilon} \varepsilon, \tag{2.19a}$$

$$S_1 \xrightarrow{a} SA, \tag{2.19b}$$

$$S_1 \xrightarrow{a} A, \tag{2.19c}$$

$$S \xrightarrow{a} SA, \tag{2.19d}$$

$$S \xrightarrow{a} A, \tag{2.19e}$$

$$A \xrightarrow{a'} S, \tag{2.19f}$$

$$A \xrightarrow{a'} \varepsilon, \tag{2.19g}$$

де a, a' — пара символів алфавіту T_1.

2.3.5 Контекстно-вільна мова L' та автомат з магазинною пам'яттю $M_{L'}$, що їй відповідає

Побудуємо тепер контекстно-вільну мову L' таку, що

$$f(L') = L, \tag{2.20}$$

та для якої ми згодом зможемо показати

$$L' = L_0 \cap D \tag{2.21}$$

Введемо позначення для послідовностей термінальних символів, що кодують додавання нетермінальних символів до стеку та вилучення їх з нього: оскільки регулярна мова L_0 породжуватиме такі послідовності, їх має породжувати і L'.

Оскільки ми вже маємо L, то знаємо і її алфавіт нетермінальних символів N. Пронумеруємо ці символи від 1 до n, де n — довжина алфавіту. Тоді для кожного $A_i \in N$ позначимо

$$x_{A_i[} = [(^i[\tag{2.22}$$

$$x_{A_i]} =])^i] \tag{2.23}$$

Тепер опишемо L'. $L = (N, T_1, P', S)$, де N і S співпадають з такими у L, а T_1 — побудований нами алфавіт.

Без обмеження загальності розглядатимемо L у нормальній формі Грейбах. Позначивши за P правила L, задамо P' таким чином:

а) Якщо P містить правило (2.2), то P' теж містить правило (2.2).

б) Для кожного правила вигляду

$$A \to a, \tag{2.24}$$

де $a \in T$;

$A \in N$;

множина P' містить правило

$$A \to x_{A]} a a', \tag{2.25}$$

де a' — парний до a символ.

в) Для кожного правила вигляду (2.1) множина P' містить таке правило:

$$A \to x_{A_1]} a a' x_{A_k[} x_{A_{k-1}[} \dots x_{A_1[} A_1 A_2 \dots A_k, \tag{2.26}$$

де a' — парний до a символ.

Єдиним виключенням є правила з S у лівій частині: вони не містять x_{S_1} на початку правої частини.

Як бачимо, з урахуванням описаного вище узагальнення щодо кількох термінальних символів на початку лівої частини правила, L' представлено у нормальній формі Грейбах. Тож можемо побудувати автомат $M_{L'}$, за наведеним вище алгоритмом.

Маємо:

$$M_{L'} = (Q, T, \Gamma, \Delta', I, S), \tag{2.27}$$

де $Q = \{q_0\}$;

$I = q_0$;

$\Gamma = N$.

Опишемо правила Δ. Знову ж-таки, єдиним станом автомата є q_0, тож просто вкажемо для кожного правила вхідний символ та стани верхівки стеку:

а) Якщо P містить правило (2.2), то $\Delta_{L'}$ містить

$$S \xrightarrow{\varepsilon} \varepsilon \tag{2.28}$$

б) Якщо P містить правило вигляду (2.24), то $\Delta_{L'}$ містить

$$A \xrightarrow{x_{A]}aa'} \varepsilon \tag{2.29}$$

в) Якщо P містить правило вигляду (2.1), то $\Delta_{L'}$ містить

$$A \xrightarrow{x_{A]}aa'x_{A_k[}x_{A_{k-1}[}\ldots x_{A_1[}} A_1A_2\ldots A_k \tag{2.30}$$

2.3.6 Регулярна мова L_0 та скінченний автомат M_{L_0}, що їй відповідає

Тепер побудуємо регулярну мову L_0. $L_0 = (N, T_1, P_0, S)$. Правила P_0 побудуємо таким чином:

а) Якщо P містить правило (2.2), то P_0 теж містить правило (2.2).

б) Для кожного правила вигляду (2.24) з P множина P_0 містить такі правила: правило (2.25) та правило

$$A \to x_{A]}aa'B \tag{2.31}$$

для кожного $B \in N\backslash\{S\}$, де a' — парний до a символ.

в) Для кожного правила вигляду (2.1) з P множина P_0 містить таке правило:

$$A \to x_{A_]} a a' x_{A_k[} x_{A_{k-1}[} \dots x_{A_1[} A_1, \tag{2.32}$$

де a' — парний до a символ.

Єдиним виключенням є правила з S у лівій частині: вони не містять $x_{S_]}$ на початку правої частини.

За описаним вище алгоритмом побудуємо скінченний автомат. $M_{L_0} = (Q, T_1, \Delta_{L_0}, I, F)$, де $Q = N \cup F$, $I = S$, а Δ_{L_0} задається таким чином:

а) Якщо P містить правило (2.2), то Δ_{L_0} містить

$$S \xrightarrow{\varepsilon} F \tag{2.33}$$

б) Якщо P містить правило вигляду (2.24), то Δ_{L_0} містить

$$A \xrightarrow{x_{A]} a a'} F \tag{2.34}$$

в) Якщо P містить правило вигляду (2.24), то для кожного $q \in Q \backslash \{S\}$ Δ_{L_0} містить правило

$$A \xrightarrow{x_{A]} a a'} q \tag{2.35}$$

г) Для кожного правила вигляду (2.1) з P Δ_{L_0} містить правило

$$A \xrightarrow{x_{A]} a a' x_{A_k[} x_{A_{k-1}[} \dots x_{A_1[}} A_1 \tag{2.36}$$

2.3.7 Доведення еквівалентності виводу $M_{L'}$ та кон'юнкції виводів M_{L_0} та M_D

Спершу розглянемо випадок, коли L допускає порожнє слово. В такому випадку, за побудовою, його допускатимуть $M_{L'}, M_{L_0}$ і M_D, а отже, виконуватиметься

$$\varepsilon \in L \iff \varepsilon \in f(L_0 \cap D) \tag{2.37}$$

Тепер без обмеження загальності вважатимемо, що L не допускає порожнього слова. В такому випадку, за побудовою, L_0 його теж не допускає, а отже,

$$\varepsilon \notin L \Longleftrightarrow \varepsilon \notin f(L_0 \cap D) \tag{2.38}$$

Зауважимо, що оскільки L не допускає порожнього слова, в усіх трьох автоматах відсутні ε-переходи.

Покажемо, що

$$\begin{aligned}(\langle q_0, wv, \{A, A_1, A_2 \dots A_n\}\rangle \underset{M_{L'}}{\vdash} \langle q_0, v, \{B_1, B_2 \dots B_m, A_1, A_2 \dots A_n\}\rangle \Longleftrightarrow \\ \Longleftrightarrow \langle A, wv\rangle \underset{M_{L_0}}{\vdash} \langle B_1, v\rangle) \wedge (\langle q_0, wv, \{A', A'_1.A'_2 \dots A'_n\}\rangle \underset{M_D}{\vdash^*} \\ \underset{M_D}{\vdash^*} \langle q_0, v, \{B'_1, B'_2 \dots B'_m, A'_1.A'_2 \dots A'_n\}\rangle),\end{aligned} \tag{2.39}$$

де $m > 0$.

Справді, нехай

$$\langle q_0, wv, \{A, A_1, A_2 \dots A_n\}\rangle \underset{M_{L'}}{\vdash} \langle q_0, v, \{B_1, B_2 \dots B_m, A_1, A_2 \dots A_n\}\rangle \tag{2.40}$$

Це означає, що для автомата $M_{L'}$ існує правило

$$A \xrightarrow{x_{A]}aa'x_{B_m[}x_{B_{m-1}[}\dots x_{B_1[}} B_1B_2 \dots B_m, \tag{2.41}$$

і виконується

$$w = x_{A]}aa'x_{B_m[}x_{B_{m-1}[} \dots x_{B_1[} \tag{2.42}$$

За алгоритмом побудови автоматів, автомат M_{L_0} містить правило

$$A \xrightarrow{x_{A]}aa'x_{B_m[}x_{B_{m-1}[}\dots x_{B_1[}} B_1, \tag{2.43}$$

а отже,

$$\langle A, wv\rangle \underset{M_{L_0}}{\vdash} \langle B_1, v\rangle \tag{2.44}$$

Розглянемо тепер, як автомат M_D, що знаходиться у стані $\langle q_0, wv, \{A', A'_1, A'_2 \dots A'_n\}\rangle$, опрацює рядок (2.42).

а) $x_{A]} \rightarrow$ прибере A' з верхівки стеку;

б) $a \rightarrow$ додасть a' до верхівки стеку;

в) $a' \rightarrow$ прибере a' з верхівки стеку;

г) $x_{B_m[}x_{B_{m-1}[}\ldots x_{B_1[} \rightarrow$ послідовно додасть до верхівки стеку $B'_m, B'_{m-1}, \ldots B'_1$

Нескладно побачити, що після цього M_D знаходитиметься у стані $\langle q_0, v, \{B'_1, B'_2 \ldots B'_m, A'_1.A'_2 \ldots A'_n\}\rangle$.

Таким чином, видно, що виконується

$$(\langle q_0, wv, \{A, A_1, A_2 \ldots A_n\}\rangle \underset{M_{L'}}{\vdash} \langle q_0, v, \{B_1, B_2 \ldots B_m, A_1, A_2 \ldots A_n\}\rangle \Rightarrow$$
$$\Rightarrow \langle A, wv\rangle \underset{M_{L_0}}{\vdash} \langle B_1, v\rangle) \wedge (\langle q_0, wv, \{A', A'_1, A'_2 \ldots A'_n\}\rangle \underset{M_D}{\vdash^*}$$
$$\underset{M_D}{\vdash^*} \langle q_0, v, \{B'_1, B'_2 \ldots B'_m, A'_1.A'_2 \ldots A'_n\}\rangle), \quad (2.45)$$

де $m > 0$.

Покажемо тепер зворотній напрямок тотожності. Нехай

$$\langle A, wv\rangle \underset{M_{L_0}}{\vdash} \langle B_1, v\rangle) \wedge (\langle q_0, wv, \{A', A'_1, A'_2 \ldots A'_n\}\rangle \underset{M_D}{\vdash^*}$$
$$\underset{M_D}{\vdash^*} \langle q_0, v, \{B'_1, B'_2 \ldots B'_m, A'_1.A'_2 \ldots A'_n\}\rangle) \quad (2.46)$$

Оскільки

$$\langle q_0, wv, \{A', A'_1, A'_2 \ldots A'_n\}\rangle \underset{M_D}{\vdash^*} \langle q_0, v, \{B'_1, B'_2 \ldots B'_m, A'_1.A'_2 \ldots A'_n\}\rangle, \quad (2.47)$$

то w має містити символи $x_{A]}, x_{B_m[}, x_{B_{m-1}[}\ldots x_{B_1[}$ у вказаному порядку та, можливо, правильні дужкові послідовності інших символів проміж себе.

Єдиною такою послідовністю, для якої у автомата M_{L_0} визначено перехід, є послідовність (2.42), що відповідає правилу (2.43). Це, в свою чергу, не суперечить переходу (2.44).

Оскільки за побудовою автомат $M_{L'}$ містить правило (2.41), то бачимо, що виконується (2.40), а отже, і

$$(\langle A, wv\rangle \underset{M_{L_0}}{\vdash} \langle B_1, v\rangle) \wedge (\langle q_0, wv, \{A', A'_1, A'_2 \dots A'_n\}\rangle \underset{M_D}{\vdash^*} \underset{M_D}{\vdash^*} \langle q_0, v, \{B'_1, B'_2 \dots B'_m, A'_1.A'_2 \dots A'_n\}\rangle \Rightarrow \Rightarrow \langle q_0, wv, \{A, A_1, A_2 \dots A_n\}\rangle \underset{M_{L'}}{\vdash} \langle q_0, v, \{B_1, B_2 \dots B_m, A_1, A_2 \dots A_n\}\rangle), \tag{2.48}$$

де $m > 0$, що і треба було довести.

Розглянемо тепер випадок $m = 0$. Покажемо, що

$$(\langle q_0, wv, \{A, A_1, A_2 \dots A_n\}\rangle \underset{M_{L'}}{\vdash} \langle q_0, v, \{A_1, A_2 \dots A_n\}\rangle \Longleftrightarrow \Longleftrightarrow \langle A, wv\rangle \underset{M_{L_0}}{\vdash} \langle A_1, v\rangle) \wedge (\langle q_0, wv, \{A', A'_1, A'_2 \dots A'_n\}\rangle \underset{M_D}{\vdash^*} \rangle \underset{M_D}{\vdash^*} \langle q_0, v, \{A'_1.A'_2 \dots A'_n\}\rangle) \tag{2.49}$$

Нехай

$$\langle q_0, wv, \{A, A_1, A_2 \dots A_n\}\rangle \underset{M_{L'}}{\vdash} \langle q_0, v, \{A_1, A_2 \dots A_n\}\rangle \tag{2.50}$$

Тоді для автомата $M_{L'}$ існує правило вигляду (2.29), і

$$w = x_{A]}aa' \tag{2.51}$$

Оскільки у автомата M_{L_0} є правило вигляду (2.35) для кожного некінцевого q, то правило

$$A \xrightarrow{x_{A]}aa'} A_1 \tag{2.52}$$

у нього теж є, відповідно, виконується

$$\langle A, wv\rangle \underset{M_{L_0}}{\vdash} \langle A_1, v\rangle) \tag{2.53}$$

Коли автомат M_D опрацьовуватиме рядок $x_{A]}aa'$, він:

а) $x_{A]} \rightarrow$ прибере A' з верхівки стеку;

б) $a \rightarrow$ додасть a' до верхівки стеку;

в) $a' \rightarrow$ прибере a' з верхівки стеку.

Таким чином, виконуватиметься

$$\langle q_0, wv, \{A', A'_1, A'_2 \dots A'_n\}\rangle \underset{M_D}{\vdash^*} \langle q_0, v, \{A'_1.A'_2 \dots A'_n\}\rangle \tag{2.54}$$

Один напрямок тотожності доведено.

Розглянемо тепер зворотній напрямок. Оскільки виконується (2.54), то w містить один символ $x_{A]}$ та, можливо, правильну дужкову послідовність з якогось або обидвох боків від нього. Оскільки ця послідовність має викликати один перехід в M_{L_0}, це послідовність вигляду $x_{A]}aa'$. Автомат $M_{L'}$ містить правило (2.29), а отже, має місце перехід (2.50), що і треба було довести.

Нарешті, розглянемо випадок, коли крок призводить до порожнього стеку. Покажемо, що

$$(\langle q_0, wv, \{A\}\rangle \underset{M_{L'}}{\vdash} \langle q_0, v, \{\}\rangle \Longleftrightarrow \langle A, wv\rangle \underset{M_{L_0}}{\vdash} \langle F, v\rangle) \wedge (\langle q_0, wv, \{A'\}\rangle \underset{M_D}{\vdash^*} \underset{M_D}{\vdash^*} \langle q_0, v, \{\}\rangle) \tag{2.55}$$

Нехай

$$\langle q_0, wv, \{A\}\rangle \underset{M_{L'}}{\vdash} \langle q_0, v, \{\}\rangle \tag{2.56}$$

Тоді для автомата $M_{L'}$ існує правило (2.29), і

$$w = x_{A]}aa' \tag{2.57}$$

Оскільки у автомата M_{L_0} є правило (2.34), то виконується

$$\langle A, wv\rangle \underset{M_{L_0}}{\vdash} \langle F, v\rangle) \tag{2.58}$$

Логіка опрацювання рядка автоматом M_D не відрізняється від попереднього випадку, тож абсолютно аналогічним чином можемо показати, що

$$\langle q_0, wv, \{A'\}\rangle \underset{M_D}{\vdash^*} \langle q_0, v, \{\}\rangle \tag{2.59}$$

В одному напрямку тотожність доведено.

Розглянемо зворотній напрямок. Оскільки виконується (2.59), то w містить один символ $x_{A]}$ та, можливо, правильну дужкову послідовність з якогось або обидвох боків від нього. Оскільки ця послідовність має викликати один перехід в M_{L_0}, це послідовність вигляду $x_{A]}aa'$. Автомат $M_{L'}$ містить правило (2.29), а отже, має місце перехід (2.56), що і треба було довести.

Отже, ми показали, що для будь-якого переходу в автоматі $M_{L'}$ існують еквівалентні переходи в M_{L_0} та M_D, що зберігають таку властивість: у $M_{L'}$ та M_D при еквівалентних переходах зберігається вміст стеку, тоді як поточний стан M_{L_0} відповідає верхньому елементу цих стеків. У випадку, коли стек порожній, автомат M_{L_0} знаходиться у кінцевому стані F.

Таким чином, якщо автомат $M_{L'}$ приймає рядок s, то і M_{L_0}, і M_D його приймають. Якщо ж рядок s одночасно приймають автомати M_{L_0} та M_D, то його приймає і $M_{L'}$.

Отже, розширене формулювання теореми Хомського-Шютценберже доведено.

2.4 Алгоритм роботи програмного продукту

Принципи проведення побудов, необхідних для доведення теореми Хомського-Шютценберже, було детально описано у попередньому розділі, тож залишається лише описати загальний алгоритм роботи програмного продукту.

а) Отримати від користувача ПП на вхід алфавіт термінальних символів T.

б) Побудувати за алфавітом T алфавіт термінальних символів T'_1.

в) Побудувати за алфавітами T та $'_1$ гомоморфізм f.

г) Побудувати за алфавітом T_1 мову Діка D та автомат з магазинною пам'яттю M_D.

д) Отримати від користувача ПП на вхід контекстно-вільну мову L.

е) Побудувати за мовою L контекстно-вільну мову L', подану у нормальній формі Грейбах, та автомат з магазинною пам'яттю $M_{L'}$.

ж) Побудувати за мовою L регулярну мову L_0, подану у нормальній формі Грейбах, та скінченний автомат M_{L_0}.

з) Очікувати від користувача на вхід слова з символів алфавіту T_1 та опрацьовувати їх, а саме:

1) Для заданого слова перевірити, чи приймають його автомати $M_{L'}, M_{L_0}, M_D$.

2) За результатами попереднього кроку вирахувати, чи приймає задане слово мова $L_0 \cap D$.

2.5 Висновки

У даному розділі було розглянуто існуючі доведення теореми Хомського-Шютценберже, наведено доведення її розширеного формулювання за допомогою автоматів з магазинною пам'яттю, а також подано алгоритми побудови конструкцій, використаних у доведенні.

Також було детально розглянуто різницю між класами контекстно-вільних та регулярних мов у контексті різниці між автоматами з магазинною пам'яттю та скінченними автоматами.

3 АРХІТЕКТУРА ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ

3.1 Обґрунтування вибору мови реалізації

Розроблений програмний продукт має, по-перше, реалізовувати описані алгоритми, а по-друге — мати зручний графічний інтерфейс. У процесі написання дипломної роботи було прийнято неочевидне рішення використати для розробки програмного продукту функціональну мову програмування.

По-перше, функціональна мова дозволяє просто описати математичні концепції та алгоритми, особливо пов’язані з обробкою рядків [11]. По-друге, у використанні функціональної мови програмування сімейства LISP, що описує програми як дужкові послідовності, для доведення теореми, що містить дужкові послідовності, є певна естетична цілісність.

Серед мов сімейства LISP можна виділити мову Racket, яку було створено на основі PLT Scheme, діалекту LISP. Головним фактором на її користь є велика кількість бібліотек для додаткової функціональності, особливо бібліотеки для створення графічного інтерфейсу та для можливості використання принципів ООП.

Таким чином, для реалізації програмного продукту було обрано функціональну мову Racket. У якості довідкового матеріалу використовувалися книги [5, 6].

3.2 Аналіз архітектури системи

3.2.1 Класи, використані у програмному продукті

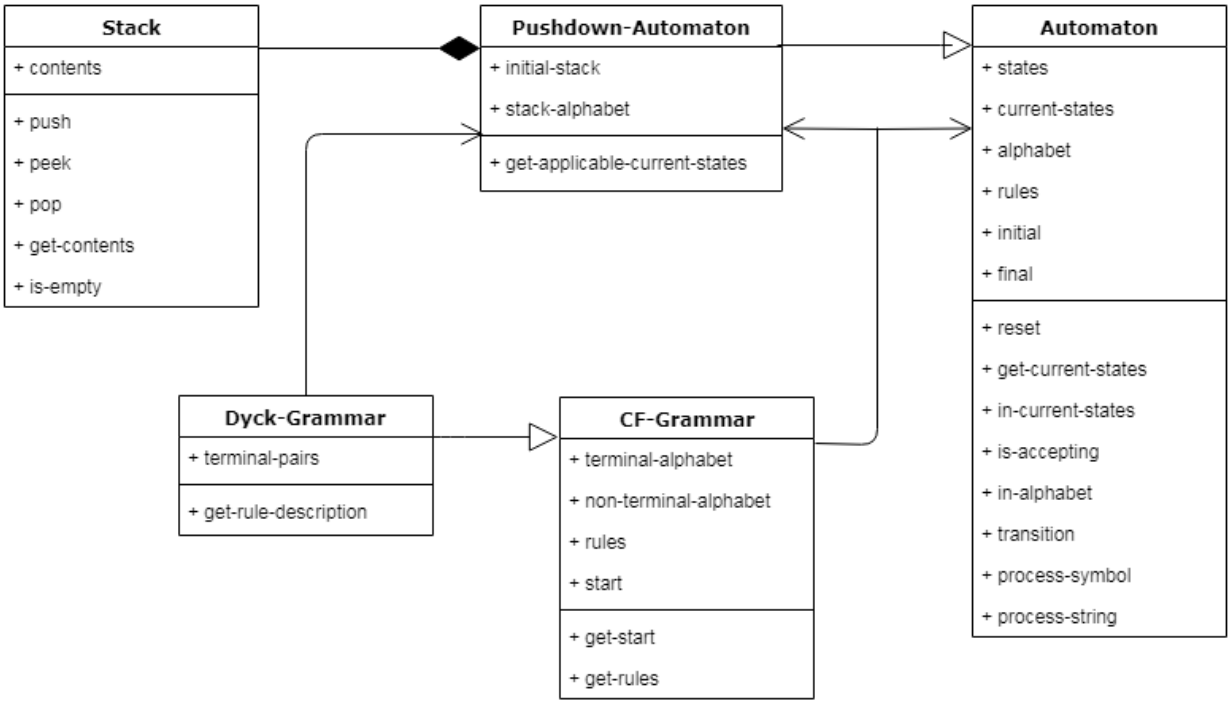

Рисунок 3.1 – UML-діаграма класів програмного продукту

Racket надає бібліотеку racket/class, що дозволяє використовувати підходи ООП, незважаючи на те, що мова є функціональною. На рис. 3.1 зображено класи, створені Для симуляції роботи автоматів у програмному продукті.

– Stack — моделює роботу стеку, дозволяє додавати елементи до стеку та вилучати їх з нього, а також переглядати верхній елемент без вилучення.
– Automaton — моделює недетермінований скінченний автомат. Зберігає довільну кількість поточних станів, дозволяє перевірити, чи якийсь з поточних станів є кінцевим, а також перейти з поточних станів до наступних за поданим символом або рядком символів.
– Pushdown-Automaton — нащадок класу Automaton, моделює недетермінований автомат з магазинною пам’яттю. Зберігає довільну кількість поточних станів, кожен з яких містить свій стек. За замовчуванням приймає рядки за порожнім стеком.

– CF-Grammar — моделює контекстно-вільну граматику, що може описувати контекстно-вільну мову (у тому числі й регулярну). Дозволяє створити автомат за поданою граматикою.
– Dyck-Grammar — моделює контекстно-вільну граматику, що описує мову Діка. Дозволяє створити таку граматику лише за набором пар символів, а також створити автомат з магазинною пам’яттю за цією граматикою.

3.2.2 Компоненти програмного продукту

Програмний продукт складається з 5 компонентів, які можна розділити на три модулі: модель, логіка та графічний інтерфейс. На рис. 3.2 зображено їхній взаємозв’язок.

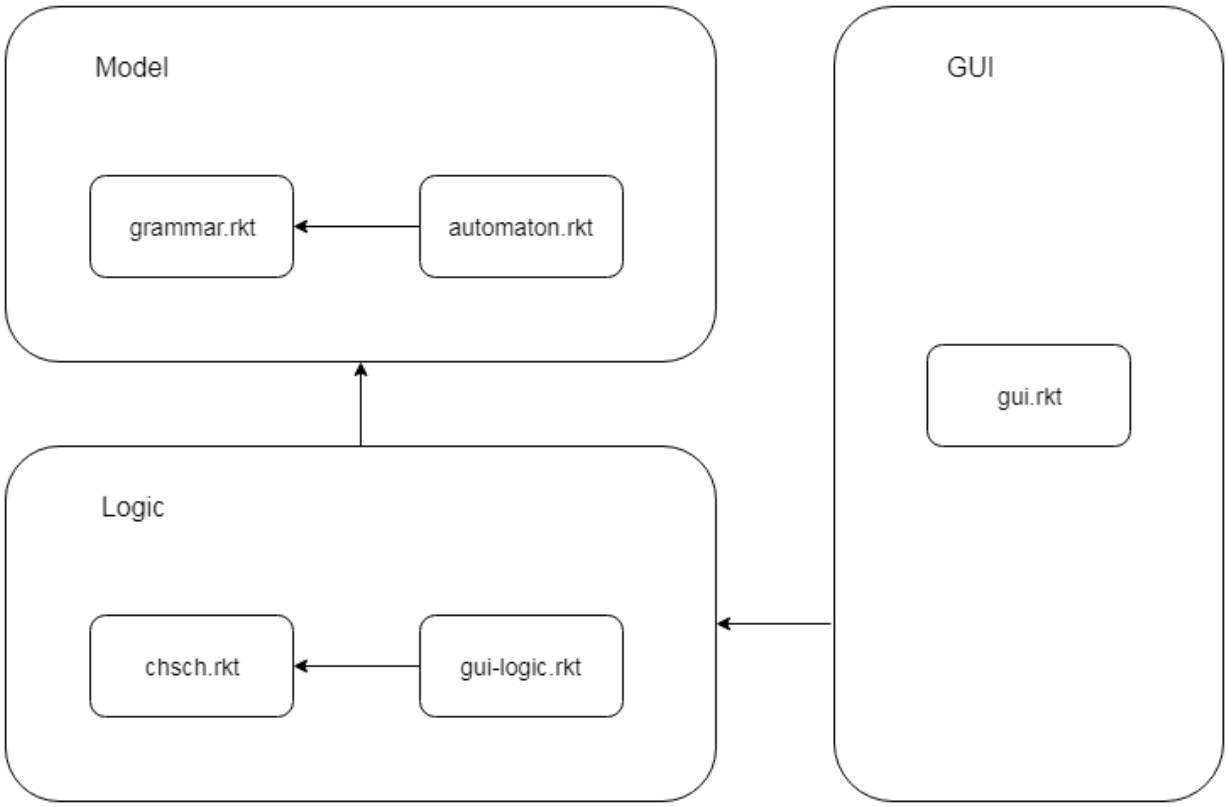

Рисунок 3.2 – Діаграма компонентів програмного продукту

Модуль «Модель» складається з описаних вище класів та задає математичні об’єкти, що використовуються у програмному продукті. Компо-

нент grammar.rkt містить класи CF-Grammar та Dyck-Grammar, а також константи, що стосуються граматик, наприклад, порожній символ. Компонент automaton.rkt містить опис та логіку роботи автоматів, а саме недетермінованих скінченних автоматів (клас Automaton) та недетермінованих автоматів з магазинною пам'яттю (клас Pushdown-Automaton). Також він містить клас Stack для опису стеку автоматів з магазинною пам'яттю. Компонент automaton.rkt залежить від компоненту grammar.rkt, оскільки в описі автоматів використовується логіка формальних мов.

Модуль «Логіка» описує внутрішню логіку програми. Компонент chsch.rkt (від Chomsky-Schützenberger) містить методи, що виконують побудови, необхідні у теоремі Хомського-Шютценберже, як-то: побудова гомоморфізму та мови Діка за алфавітом термінальних символів, побудова регулярної мови за контекстно-вільною та ін.

Компонент gui-logic.rkt поєднує логіку формальних мов у програмному продукті з графічним інтерфейсом. Він містить методи, що опрацьовують поля введення інформації та створюють об'єкти формальних мов з рядків введеної інформації. Також він викликає методи chsch.rkt, що виконують необхідні операції, при обробці натискань на кнопки інтерфейсу.

Обидва ці компоненти залежать від компонентів з модулю «Модель».

Компонент «Графічний інтерфейс» описує, власне, графічний інтерфейс програми за допомогою бібліотеки racket/gui. Він залежить від компоненту gui-logic.rkt, що займається обробкою дій користувача.

3.2.3 Керівництво користувача

Розроблений програмний продукт, «Interactive Chomsky-Schutzenberger Proof», дозволяє симулювати кроки побудови мов та автоматів, що використовуються у конструктивному доведенні теореми Хомського-Шютценберже, та перевірити еквівалентність побудов на будь-якому введеному рядку.

Інтерфейс організовано за допомогою вкладок, кожна з яких позначає певний крок побудови. Під час введення інформації у будь-який момент можна повернутися на одну з попередніх вкладок та змінити вже введену інформацію, що покличе за собою оновлення виконаних побудов. Остання вкладка дозволяє перевірити побудови за допомогою введеного рядка.

На рис. 3.3 зображено першу вкладка інтерфейсу, «Alphabet». Вона призначена для введення алфавіту термінальних символів. Символи можуть складатися з однієї або кількох літер та мають бути розділені пропусками.

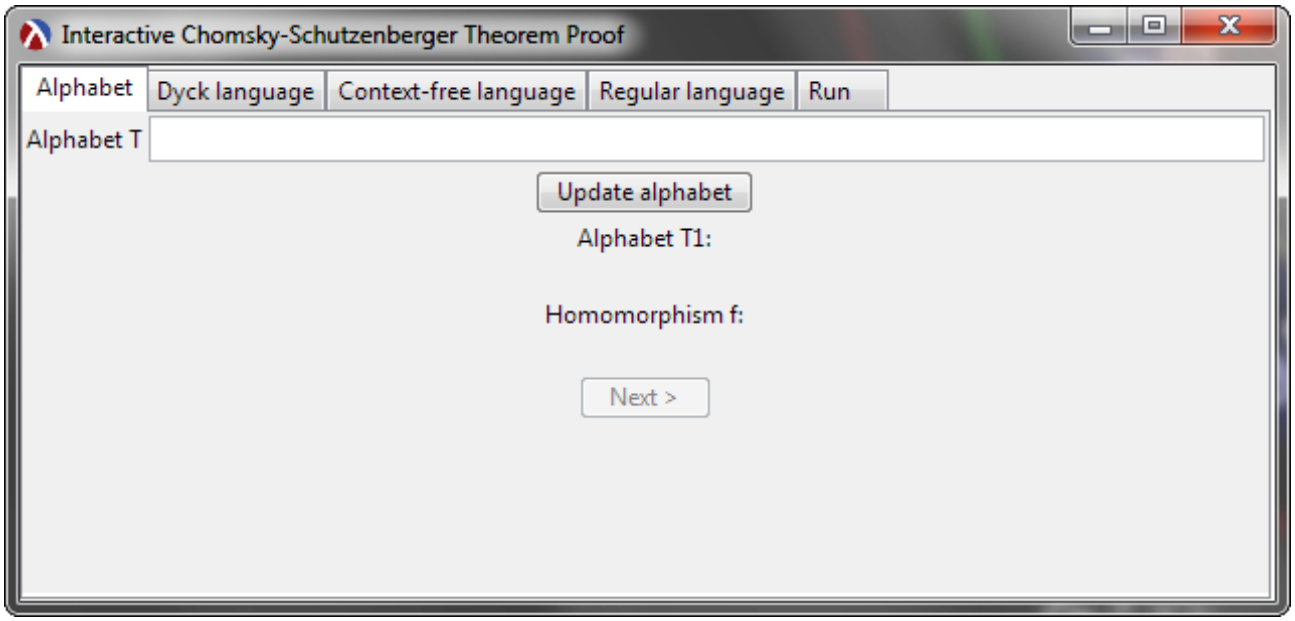

Рисунок 3.3 – Вкладка «Алфавіт»

Після введення символів та натискання на кнопку «Update alphabet» («Оновити алфавіт») програма виконує побудову алфавіту T_1 та гомоморфізму f, відображає їх, а також дозволяє перейти на наступну вкладку за домопогою кнопки «Next». Можливий результат зображено на рис. 3.4.

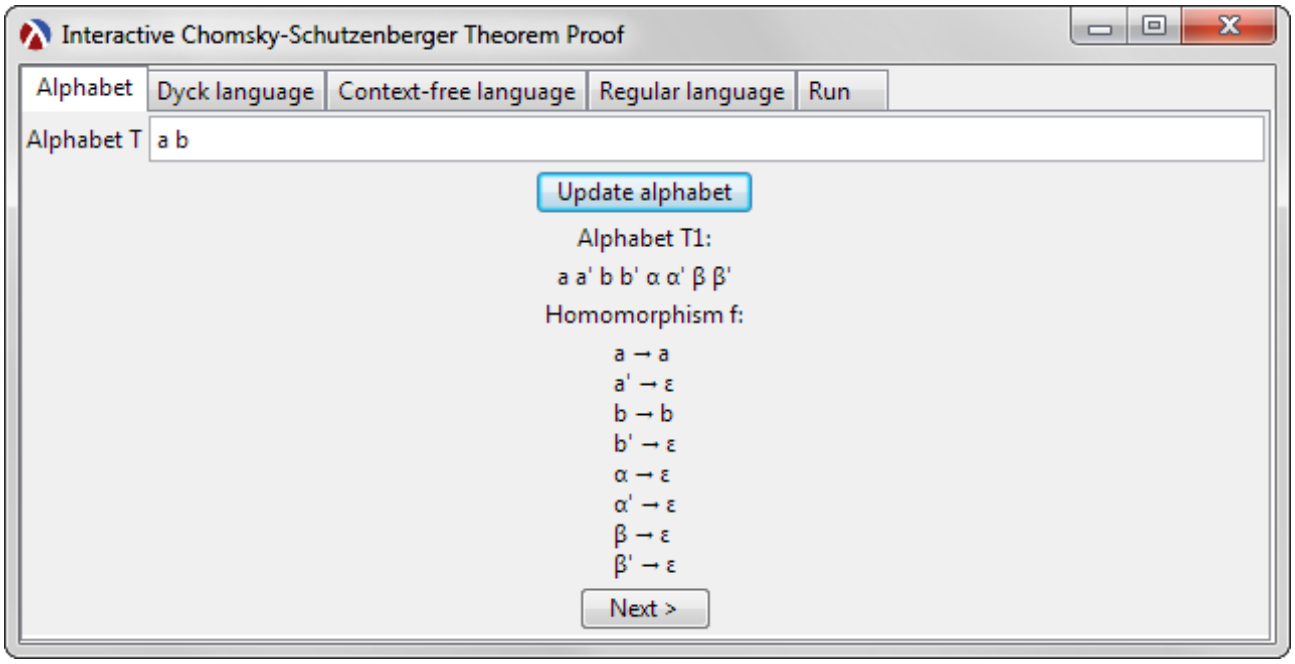

Рисунок 3.4 – Вкладка «Алфавіт» після введення даних

Наступна вкладка, «Dyck language», містить правила мови Діка та правила переходу автомату, побудованого за цією мовою. Ці правила вираховуються самостійно за алфавітом T. На рис. 3.5 зображено можливий її вигляд.

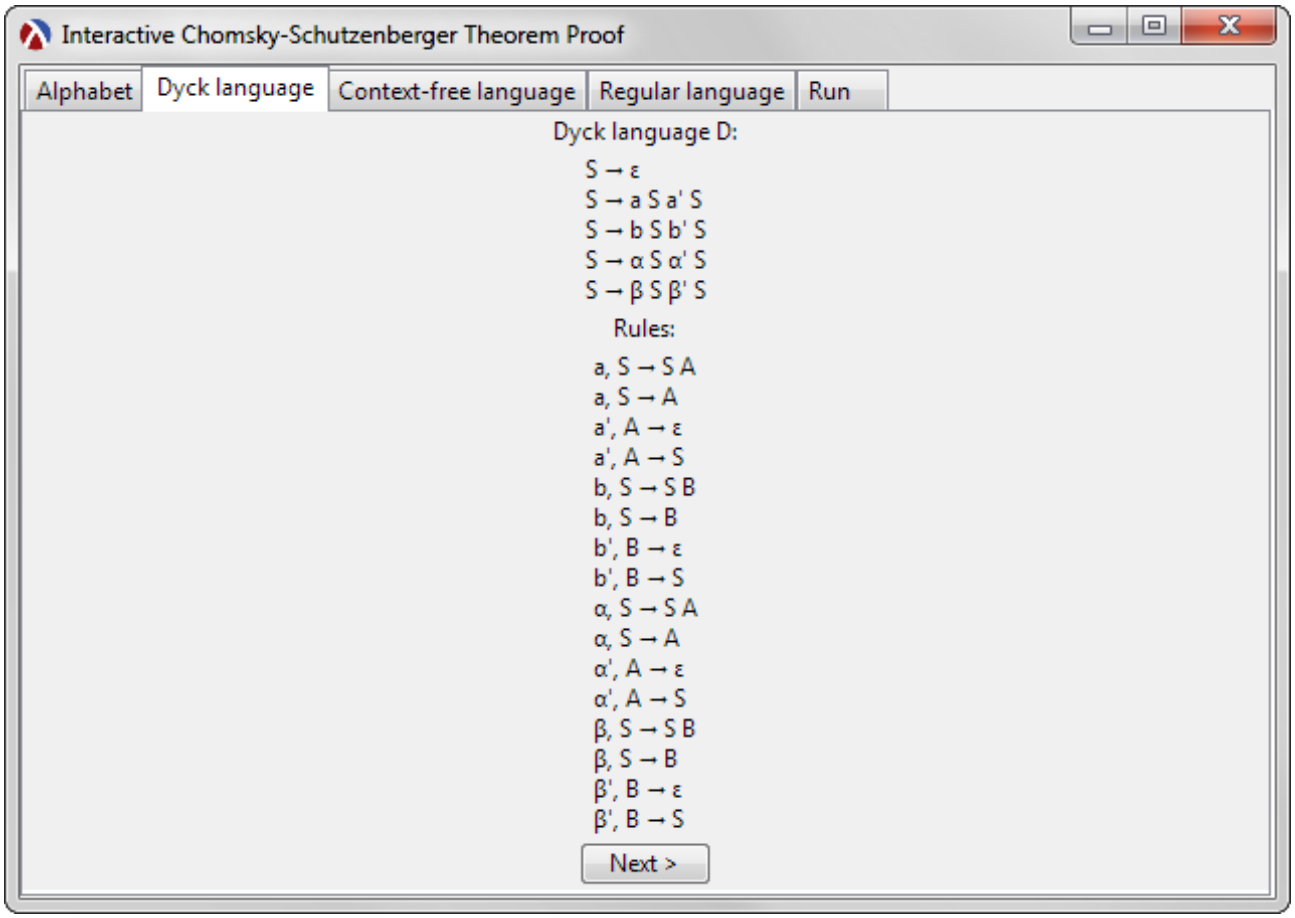

Рисунок 3.5 – Вкладка «Мова Діка»

Наступна вкладка, «Context-free language», дозволяє ввести контекстно-вільну мову L. Її початковий вигляд зображено на рис. 3.6.

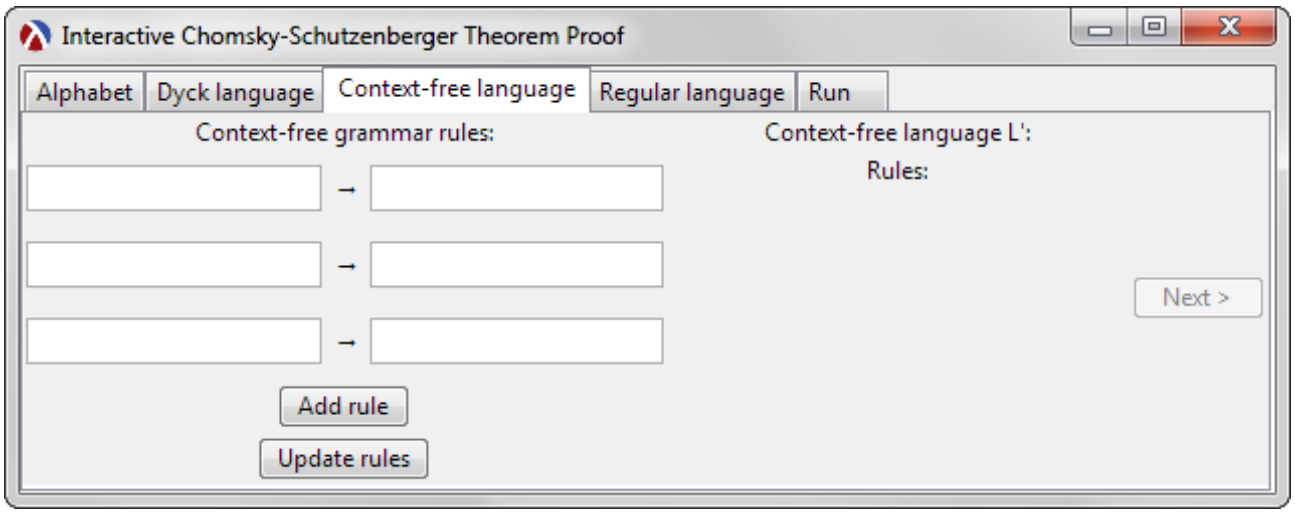

Рисунок 3.6 – Вкладка «Контекстно-вільна мова»

З лівого боку можна ввести правила мови. Ліва частина кожного правила може містити лише один нетермінальний символ, тоді як права частина може містити один або декілька термінальних або нетермінальних символів, або бути пустою.

Якщо правил має бути більша кількість, потрібно натиснути кнопку «Add rule» («Додати правило»), тоді з’явиться ще один рядок для вводу правил.

Кнопка «Update rules» («Оновити правила») записує подані правила та створює на їх основі контекстно-вільну мову L' та регулярну мову L_0, а також відповідні автомати. На рис. 3.7 зображено можливий вигляд вкладки після введення 5 правил та натискання кнопки «Update rules».

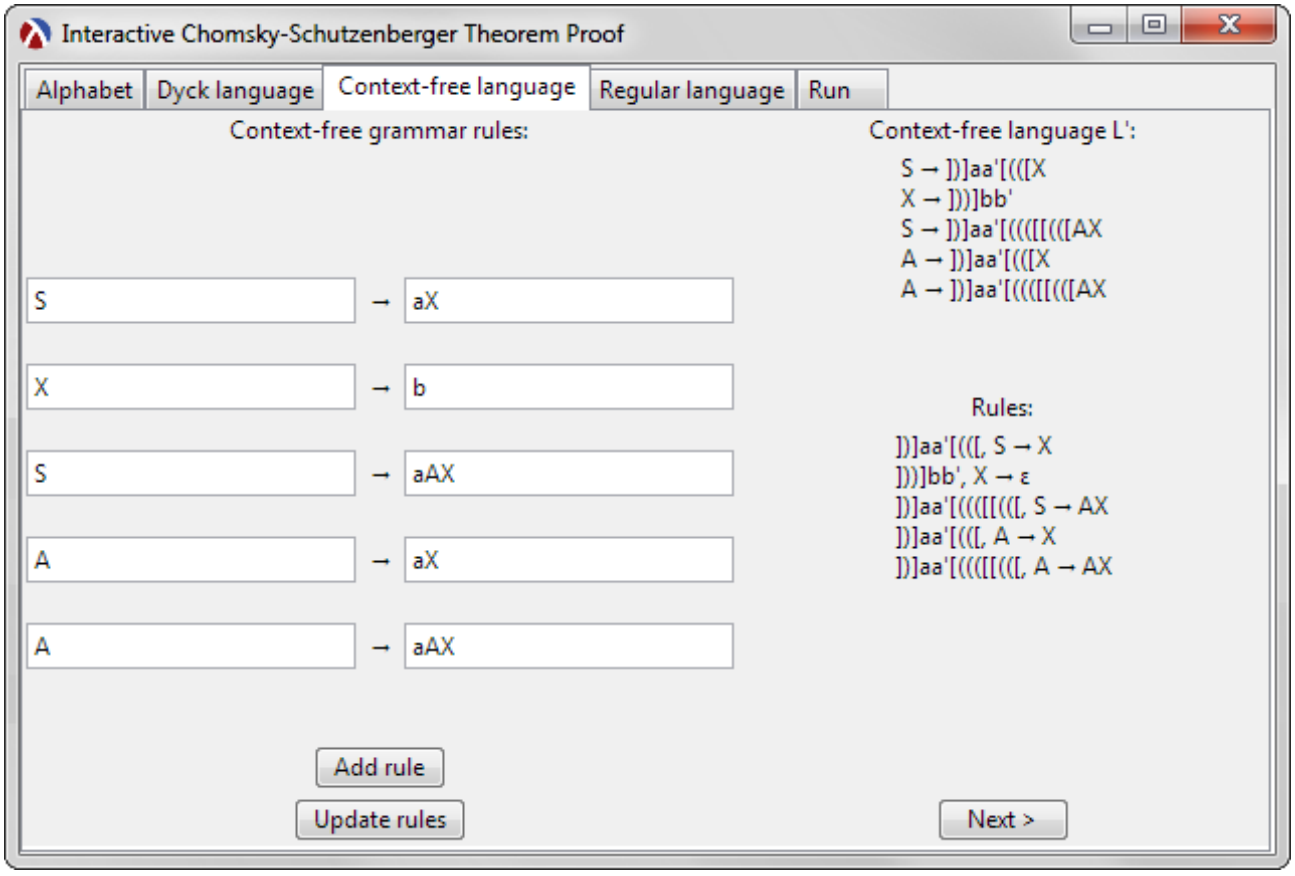

Рисунок 3.7 – Вкладка «Контекстно-вільна граматика» після введення даних

Наступна вкладка, «Regular language», відображає регулярну мову, щойно створену за наданою контекстно-вільною. На рис. 3.8 зображено її можливий вигляд.

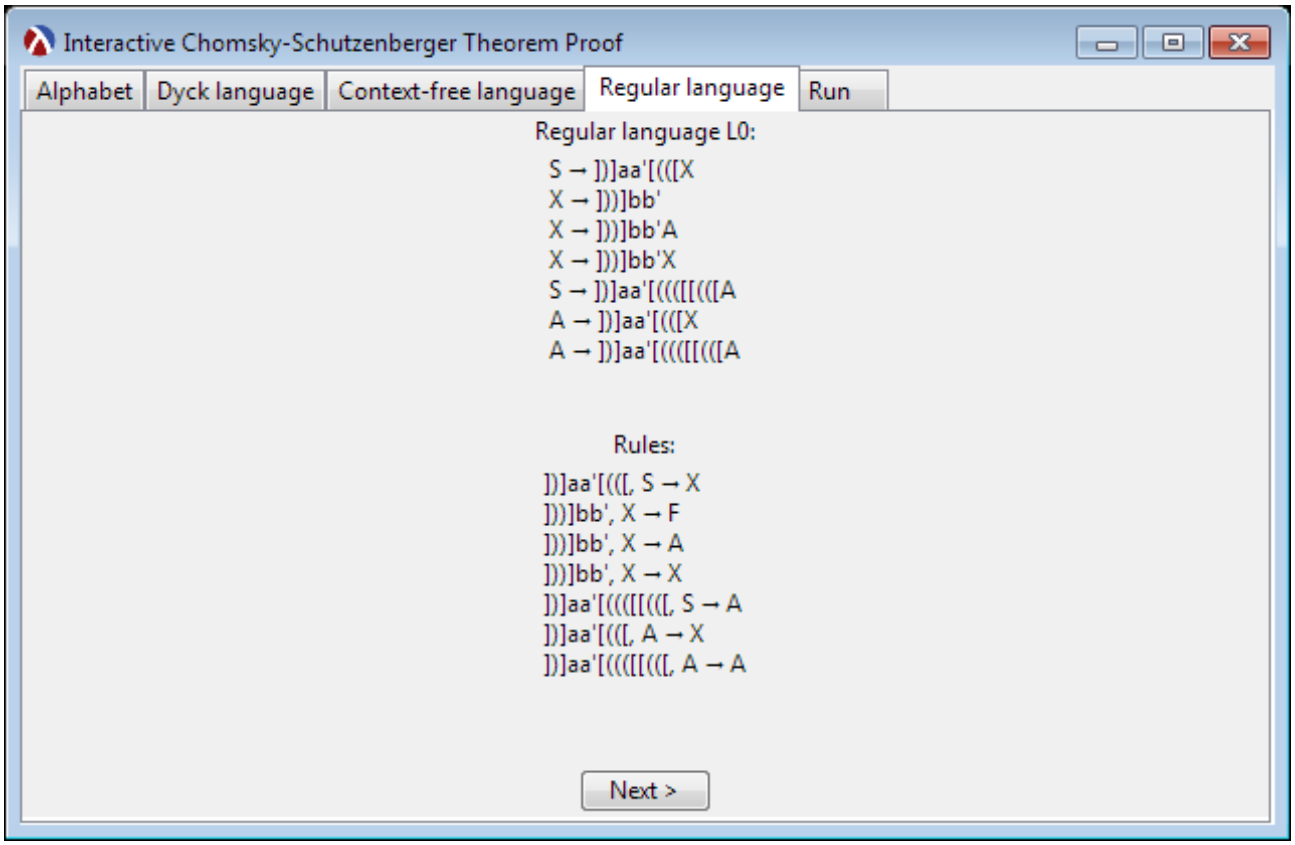

Рисунок 3.8 – Вкладка «Регулярна мова»

Наступна вкладка, «Run», дозволяє взаємодіяти з побудованими автоматами: введений рядок буде використано як вхідні дані для усіх трьох автоматів, після чого буде відображено інформацію, чи приймається цей рядок автоматами $M_{L'}, M_{L_0}, M_D$, а також кон'юнкцію результатів для L_0 і D. За твердженням теореми, результати для мови L' і для перетину L_0 і D мають бути однаковими. На рис. 3.9 зображено стан вкладки до вводу вхідних даних.

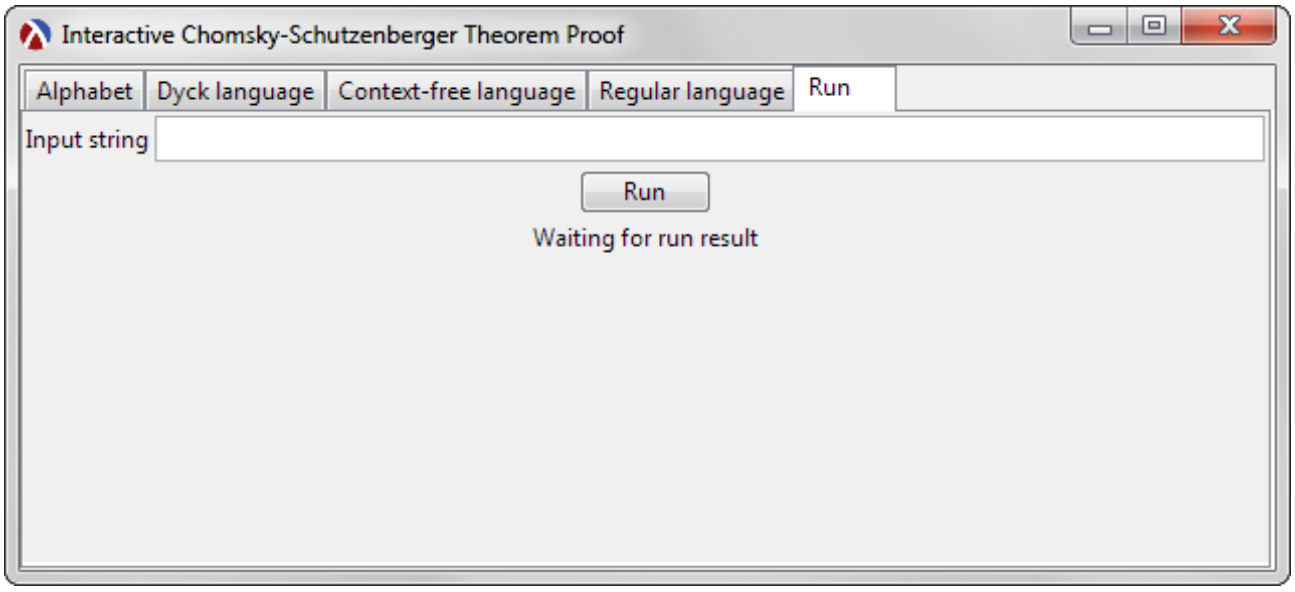

Рисунок 3.9 – Вкладка «Запуск»

До натискання кнопки «Run» («Запуск») відображається надпис, що свідчить про очікування запуску. Після натискання кновки «Run» відображаються результати, в яких «pass» свідчить про те, що рядок було розпізнано автоматом, а «fail» — що його не було розпізнано.

3.3 Аналіз результатів роботи програми

Для прикладу було розглянуто контекстно-вільну мову, що задає рядки вигляду $a^n b^n$. Попередні рисунки демонструють етапи побудови автоматів для цієї мови за алгоритмом доведення теореми Хомського-Шютценберже. Нижче наведено приклади результатів роботи програми для різних вхідних рядків.

У випадку, коли рядок розпізнається автоматами M_{L_0} та M_D, він розпізнається автоматом $M_{L'}$ (див. рис. 3.10).

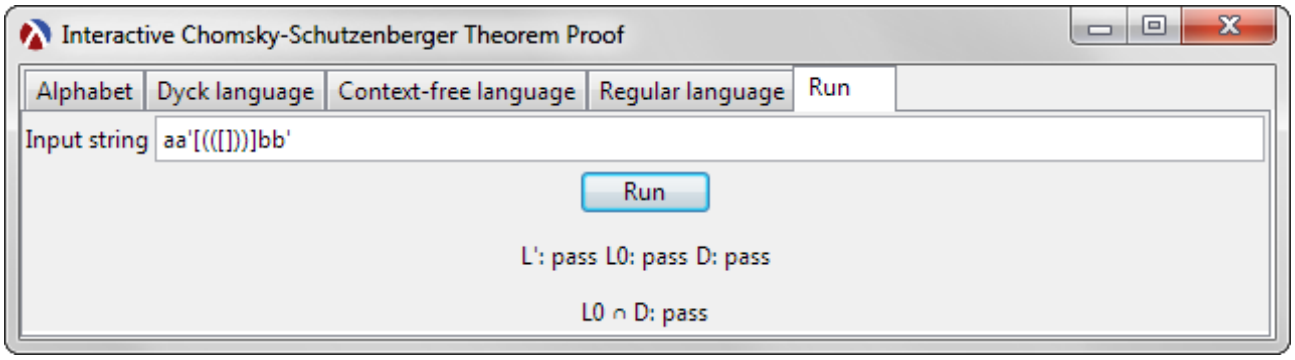

Рисунок 3.10 – Приклад: рядок належить до мови L' та до $L_0 \cap D$

У випадку, коли рядок розпізнається автоматом M_{L_0}, але не M_D, він не розпізнається автоматом $M_{L'}$ (див. рис. 3.11).

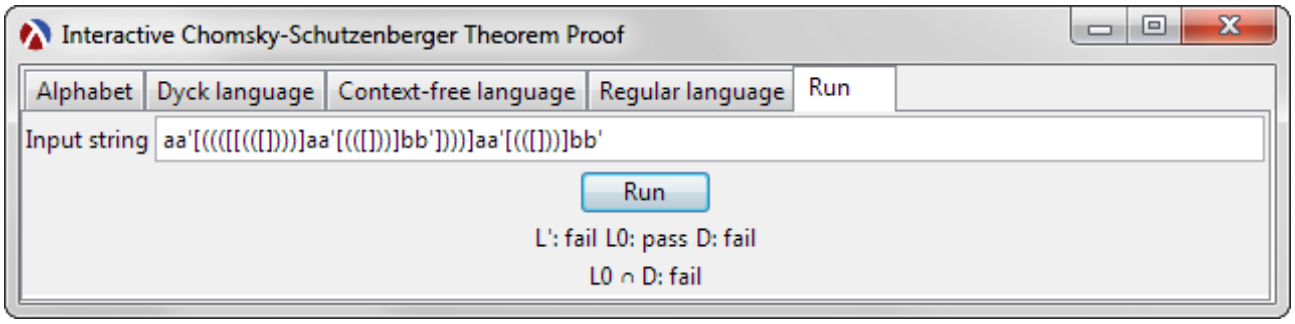

Рисунок 3.11 – Приклад: рядок належить до мови L_0, але не D та L'

У випадку, коли рядок розпізнається автоматом M_D, але не M_{L_0}, він не розпізнається автоматом $M_{L'}$ (див. рис. 3.12).

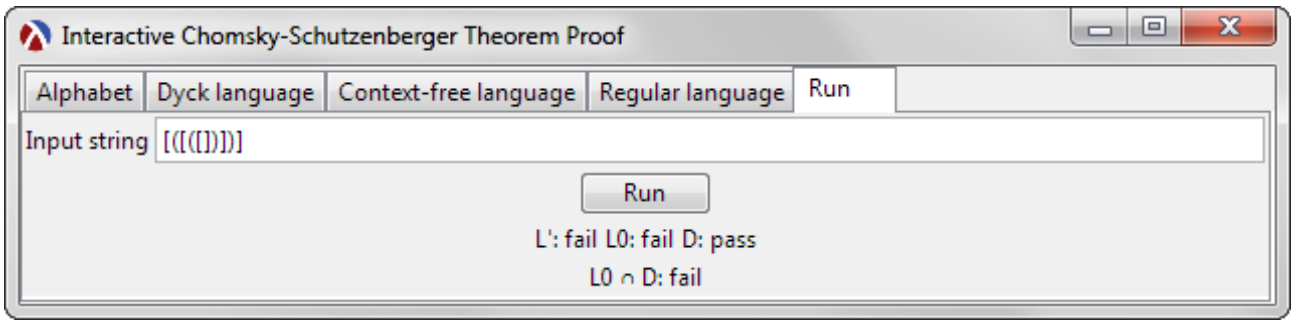

Рисунок 3.12 – Приклад: рядок належить до мови D, але не L_0 та L'

Можна бачити, що теорема Хомського-Шютценберже підтверджується експериментальними перевірками.

3.4 Висновки

У даному розділі було описано розроблений програмний продукт, класи, використані у ньому, та модулі, з яких він складається. Також було продемонстровано графічний інтерфейс та надано керівництво користувача для роботи з програмним продуктом.

ВИСНОВКИ

У роботі було продемонстровано доведення теореми Хомського-Шютценберже в розширеному формулюванні за допомогою автоматів з магазинною пам’яттю та детально розглянуто зв’язок між формальними мовами, автоматами, магазинною пам’яттю та правильними дужковими послідовностями.

Робота дозволяє з нового ракурса поглянути на зв’язок між класами формальних мов та пов’язати його зі зв’язком між класами автоматів з використанням стеку як абстракції для правильних дужкових послідовностей.

Також було розроблено програмний продукт, що дозволяє автоматично виконувати побудови, використані у доведенні теореми Хомського-Шютценберже, та перевіряти кінцевий результат.

Роботу та програмний продукт можна використовувати для демонстрації внутрішньої структури формальних мов та автоматів. Доведення викладено з використанням математичного апарату, зрозумілого студентам молодших курсів ВНЗ, а програмний продукт є способом наочно продемонструвати ідею теореми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

[1] P Braffort. Computer programming and formal systems. Amsterdam: North-Holland Pub. Co, 1963. 161 с.

[2] Stanley Burris. A Course in universal algebra: the millenium edition. Ontario: S. Burris и H.P. Sankappanavar, 2012. 276 с.

[3] Noam Chomsky. «On certain formal properties of grammars». В: Information and Control 2.2 (1959), с. 137—167.

[4] Adrian Dediu. Language and automata theory and applications : 4th international conference, LATA 2010, Trier, Germany, May 24-28, 2010 : proceedings. Berlin New York: Springer, 2010, с. 345—358.

[5] Matthias Felleisen. How to design programs: an introduction to programming and computing. Cambridge, Massachusetts: MIT Press, 2018. 792 с.

[6] Matthias Felleisen. Realm of Racket : learn to program, one game at a time. San Francisco: No Starch Press, 2013. 320 с.

[7] Seymour Ginsburg. The Mathematical Theory of Context-Free Languages. New York, NY, USA: McGraw-Hill, Inc., 1966. 258 с.

[8] Dmitry Golubenko. «Proving Parikh's theorem using Chomsky-Schutzenberger theorem». В: CoRR abs/1710.09612 (2017), с. 6—11.

[9] Sheila Greibach. «A new normal-form theorem for context-free phrase structure grammars». В: Journal of Symbolic Logic 34.4 (1970), с. 658—658.

[10] Hermann Gruber, Jonathan Lee и Jeffrey Shallit. «Enumerating regular expressions and their languages». В: CoRR abs/1204.4982 (2004), с. 17—24.

[11] Max Hailperin. Concrete abstractions : an introduction to computer science using Scheme. Pacific Grove: Brooks/Cole Pub. Co, 1999. 670 с.

[12] John Hopcroft. Introduction to automata theory, languages, and computation. Reading, Mass: Addison-Wesley, 1979. 647 с.

[13] Werner Kuich. Semirings, automata, languages. Berlin New York: Springer-Verlag, 1986. 480 c.

[14] Rohit J. Parikh. «On Context-Free Languages». В: J. ACM 13.4 (1966), с. 570—581.

[15] Grzegorz Rozenberg. Handbook of formal languages. Berlin New York: Springer, 1997, с. 322.

[16] Michael Sipser. Introduction to the theory of computation. Boston: PWS Pub. Co, 1997, с. 101—114.

[17] Пентус М. Р. Пентус А. Е. Теория формальных языков: Учебное пособие. Изд-во ЦПИ при механико-математическом ф-те МГУ, 2004. 80 с.

[18] Е.Ю Полищук, И.В. Бардадым и Д.К. Левин. «Альтернативное доказательство теоремы Хомского-Шютценберже». В: Системный анализ и информационные технологии. материалы 14-й Международной научно-технической конференции SAIT 2012. Под ред. Н. Панкратова. Киев: УНК “ИПСА” НТУУ “КПИ”, 2012, с. 109—110.

ДОДАТОК А

Лістинг програмного продукту

Модуль «grammar.rkt», що описує моделі формальних граматик.

```
#lang racket

(provide (struct-out rule))
(provide epsilon)
(provide dyck-grammar%)

; empty symbol
(define epsilon ε"")

; premise and conclusion are both lists
(struct rule (premise conclusion)
  #:transparent)

(define cf-grammar%
  (class object%
    (super-new)
    (init-field [terminal-alphabet '()]
                [non-terminal-alphabet '()]
                [rules '()]
                [start epsilon])
    (define/public (get-start) start)
    (define/public (get-rules) rules)
  ))

; terminal alphabet is n types of pairs/brackets: a list of pairs
(define dyck-grammar%
  (class cf-grammar%
    (init terminal-pairs)
    (define start "S")
    (define non-terminal-alphabet (list start))
    (define rules
            ; S → epsilon
      (cons (rule start (list epsilon))
            ; S → a_i S b_i S for every pair of brackets
            (map λ( (bracket-pair)
                    (rule start (list (first bracket-pair) start (last bracket-pair) start)))
                 terminal-pairs)
            )
      )
    (super-new
     [start start]
     [terminal-alphabet (flatten terminal-pairs)]
     [non-terminal-alphabet non-terminal-alphabet]
     [rules rules])
```

```
  (define/public (get-rule-description)
    (map λ( (r) (string-append (rule-premise r) " → " (string-join (rule-conclusion r)))) rules))
  )
)
```

Модуль «automaton.rkt», що описує моделі автоматів.

```
#lang racket
(require racket/class)
(require ”grammar.rkt”)

(provide pushdown-automaton%)
(provide (struct-out automaton-premise))
(provide (struct-out pda-premise))
(provide (struct-out pda-conclusion))

; structs
(struct automaton-premise (state input)
  #:transparent)

(struct pda-premise automaton-premise (top))

(struct pda-conclusion (state top) #:transparent)

(struct pda-state (state stack) #:transparent)

; for our purposes, empty stack returns the ”epsilon” empty symbol
(define stack%
  (class object%
    (super-new)
    (init-field [contents '()])
    ; only push non-epsilon
    (define/public (push x) (
                             if (equal? x epsilon)
                                #f
                                (set! contents (cons x contents))
                                )
      )
    (define/public (peek)
      (if (is-empty)
          epsilon
          (car contents))
      )
    (define/public (pop)
      (if (is-empty)
          epsilon
          (let ((result (car contents)))
            (set! contents (cdr contents))
            result)
          )
      )
    (define/public (is-empty) (empty? contents))
    (define/public (get-contents) contents)
    )
  )

; non-deterministic
```

```
(define automaton%
  (class object%
    (super-new)
    (init-field [states '()]
                [alphabet '()]
                [rules '()]
                [initial #f]
                [final #f]
                )
    (init-field [current-states (list initial)])
    (define/public (reset) (set! current-states (list initial)))

    (define/public (get-current-states) current-states)

    (define/public (in-current-states q) (index-of current-states q))

    (define/public (is-accepting) (in-current-states final))

    (define/public (in-alphabet x)
      (index-of alphabet x))

    ; returns a conclusion reachable by rule r, if currently applicable
    (define/public (transition r)
      (if (in-current-states (automaton-premise-state (rule-premise r)))
          (rule-conclusion r)
          '()
          )
      )

    ; process next symbol (maybe multiple characters), return unprocessed string
    (define/public (process-symbol x)
      (if (in-alphabet x)
          ; process all rules for which premise state is in current states
          ; and symbol is the current symbol
          ; BFS, essentially
          (set! current-states
                ; get all states the current state and input yield
                (flatten (map λ( (r) (transition r))
                              ; all rules that have current symbol as input
                              (filterλ
                               ( (r) (string-prefix? x (automaton-premise-input (rule-premise r))))
                               rules)
                              )
                         )
                )
          #f
          )
      )

    ; string as a list of symbols
    (define/public (process-string s)
```

```
      ; process all symbols
      (map λ( (x) (process-symbol x)) s)
      (is-accepting)
      )
    )
  )

(define pushdown-automaton%
  (class automaton%

    (super-new)
    (inherit-field initial)
    (init-field initial-stack)
    (inherit-field current-states)
    (define/override (reset)
      (set! current-states (list (pda-state initial (new stack%))))
      (send (pda-state-stack (car current-states)) push initial-stack)
      )
    (reset)

    (init-field {stack-alphabet '()})

    (inherit-field rules)
    (inherit-field final)
    (inherit in-alphabet)
    (inherit process-string)

    ; PDA is accepting iff its stack is empty
    (define/override (is-accepting)
      (not (empty?
            (filter λ( (st) (send (pda-state-stack st) is-empty))
                    current-states)
            ))
      )

    (define/public (get-applicable-current-states premise)
      (filterλ
       ( (st)
         (equal?
          (pda-premise-top premise)
          (send (pda-state-stack st) peek)))
       (filterλ
        ( (st)
          (equal?
           (automaton-premise-state premise)
           (pda-state-state st)))
        current-states)
       )
      )

    ; returns a conclusion reachable by rule r, if currently applicable, while adding to the stack
```

```
  (define/override (transition r)
    (map λ( (st) (
                  pda-state
                  (pda-conclusion-state (rule-conclusion r))
                  (new stack%
                       [contents
                        (cons
                         (pda-conclusion-top (rule-conclusion r))
                         (cdr (get-field contents (pda-state-stack st))))]
                       )))
         (get-applicable-current-states (rule-premise r)))
    )
  )
)
```

Модуль «chsch.rkt», що описує побудови, необхідні у ході доведення теореми Хомського-Шютценберже.

```
#lang racket

(require ”grammar.rkt” ”automaton.rkt”)

(provide alphabet paired-alphabet homomorphism dyck-grammar dyck-automaton)
(provide alphabet-update)
(provide get-homomorphism-description)
(provide cf-update)

(define alphabet '())
(define paired-alphabet '())
(define dyck-grammar '())
(define dyck-automaton '())

; alphabet

; update alphabet, paired alphabet, Dyck grammar
(define (alphabet-update alph)
  {println alph}
  (set! alphabet alph)
  (set! paired-alphabet (create-paired-alphabet))
  (set! dyck-grammar (new dyck-grammar% [terminal-pairs paired-alphabet]))
  (set! dyck-automaton (create-dyck-automaton))
  )

(define (create-paired-alphabet)
  (append
   (map λ( (c) (list c (string-append c ”'”))) alphabet)α
   '((”” α”'”) β(”” β”'”))
   ))

; homomorphism f
(define (homomorphism c)
    (if (index-of alphabet c)
        c
        epsilon))

(define (get-homomorphism-description)
  (flatten
   (mapλ
    ( (c)
      (string-append c ” → ” (homomorphism c)))
    (flatten paired-alphabet))))

; Dyck grammar

(define (create-dyck-automaton)
  (let ((rules
```

```
          (flatten
           (map
            ; a, S → SA
            ; a, S → A
            ; a', A →ε
            ; a', A → Sλ
            ( (symbol-pair)(apply λ( (former latter)
                                    (let ((n-former (string-upcase former)))
                                      (list
                                       (rule (pda-premise "q0" former "S")
                                             (pda-conclusion "q0" (list "S" n-former)))
                                       (rule (pda-premise "q0" former "S")
                                             (pda-conclusion "q0" (list n-former)))
                                       (rule (pda-premise "q0" latter n-former)
                                             (pda-conclusion "q0" (list epsilon)))
                                       (rule (pda-premise "q0" latter n-former)
                                             (pda-conclusion "q0" '("S")))
                                       ))) symbol-pair))
            paired-alphabet))))
      (new pushdown-automaton%
           [alphabet (flatten paired-alphabet)]
           [stack-alphabet (cons "S" (map car paired-alphabet))]
           [initial-stack "S"]
           [states '("q0")]
           [initial "q0"]
           [rules rules]
           )))

; Context-free language

(define (cf-update rules)
  (println rules))
```

Модуль «gui-logic.rkt», що забезпечує зв’язок між графічним інтерфейсом та внутрішньою логікою програмного продукту.

```
#lang racket/gui

(require ”chsch.rkt” ”grammar.rkt” ”automaton.rkt”)

(provide switch-tab)
(provide alphabet-process alphabet-update-values)
(provide cf-add-rule-input cf-process run-automata)

; tab management

(define (switch-tab tab-panel next-tab next-number)
  (send tab-panel set-selection next-number)
  (send tab-panel change-childrenλ
        ( (children) (list next-tab)))
  (send next-tab show #t))

; Alphabet tab logic

(define (alphabet-process alphabet-field)
  (alphabet-update (string-split (send alphabet-field get-value)))
  )

(define (alphabet-update-values
         paired-alphabet-message
         homomorphism-message
         dyck-grammar-message
         dyck-rules-message
         )
  (send paired-alphabet-message set-label (create-paired-alphabet-message))
  (send homomorphism-message set-label (create-homomorphism-message))
  (send dyck-grammar-message set-label (create-dyck-grammar-message))
  (send dyck-rules-message set-label (create-pda-rules-message dyck-automaton))
  )

(define (create-paired-alphabet-message)
  (string-join (flatten paired-alphabet)))

(define (create-homomorphism-message)
  (string-join (get-homomorphism-description) ”\n”))

; Dyck language tab logic

(define (create-dyck-grammar-message)
  (string-join (send dyck-grammar get-rule-description) ”\n”))

(define (create-pda-rules-message automaton)
  (string-join
   (map λ( (rule)
```

```
          (string-append
           (automaton-premise-input (rule-premise rule))
           ", "
           (pda-premise-top (rule-premise rule))
           " → "
           (string-join (pda-conclusion-top (rule-conclusion rule)))))
        (get-field rules automaton))
   "\n"
   )
  )

; Context-free language tab logic

(define (cf-add-rule-input cf-parent)
  (define rule (new horizontal-pane% [parent cf-parent]))
  (new text-field% [parent rule]
       [label #f])
  (new message% [parent rule]
       [label " → "])
  (new text-field% [parent rule]
       [label #f]))

(define (cf-process rule-list)
  (cf-update
   (mapλ
    ( (row)
      (let
          ((inputs
            (filterλ
             ( (child) (is-a? child text-field%))
             (send row get-children))
            ))
        (rule (send (first inputs) get-value) (send (second inputs) get-value))
        ))
      (send rule-list get-children))
   )
  )

; Regular language tab logic

(define (create-retular-grammar-message)
  (string-join (send regular-grammar get-rule-description) "\n"))

(define (create-fsa-rules-message automaton)
  (string-join
   (map λ( (rule)
          (string-append
           (automaton-premise-input (rule-premise rule))
           ", "
           (pda-premise-top (rule-premise rule))
           " → "
```

```
           (string-join (pda-conclusion-top (rule-conclusion rule)))))
         (get-field rules automaton))
    "\n"
    )
  )

; Run tab logic

(define (run-automata)
  ((and
     (send dyck-automaton is-accepting)
     (send cf-automaton is-accepting)
     (send r-automaton is-accepting))
       ))
```

Модуль «gui.rkt», що описує графічний інтерфейс програмного продукту.

```
#lang racket/gui

(require ”gui-logic.rkt”)

; main GUI
(define main-frame
  (new frame%
       [label ”Interactive Chomsky-Schutzenberger Theorem Proof”]
       [height 480]
       [stretchable-height 480]
       [width 640]))

(define tab-panel
  (new tab-panel% [parent main-frame]
       [choices ’(”Alphabet” ”Dyck language” ”Context-free language” ”Regular language” ”Run”)]
       [callbackλ
        ( (tp event)
          (case (send tp get-item-label (send tp get-selection))
            [(”Alphabet”)
             (send tp change-childrenλ
                   ( (children) (list alphabet-tab)))]
            [(”Dyck language”)
             (send tp change-childrenλ
                   ( (children) (list dyck-language-tab)))]
            [(”Context-free language”)
             (send tp change-childrenλ
                   ( (children) (list cf-language-tab)))]
            [(”Regular language”)
             (send tp change-childrenλ
                   ( (children) (list r-language-tab)))]
            [(”Run”)
             (send tp change-childrenλ
                   ( (children) (list run-tab)))]
            ))]))

; Alphabet tab
(define alphabet-tab
  (new vertical-panel% [parent tab-panel]))

(define alphabet-field
  (new text-field% [parent alphabet-tab]
       [label ”Alphabet T”]
       ))

(new button% [parent alphabet-tab]
     [label ”Update alphabet”]
     [callback λ( (button event)
                 (alphabet-process alphabet-field)
                 (alphabet-update-values
                  paired-alphabet-message
```

```
                 homomorphism-message
                 dyck-grammar-message
                 dyck-rules-message
                 )
                (send alphabet-next enable #t)
                )])

(new message% [parent alphabet-tab]
       [label "Alphabet T1:"])

(define paired-alphabet-message
  (new message% [parent alphabet-tab]
       [label ""]
       [auto-resize #t]))

(new message% [parent alphabet-tab]
       [label "Homomorphism f:"])

(define homomorphism-message
  (new message% [parent alphabet-tab]
       [label ""]
       [auto-resize #t]))

(define alphabet-next
  (new button% [parent alphabet-tab]
       [label "Next >"]
       [enabled #f]
       [callbackλ
        ( (button event)
          (switch-tab tab-panel dyck-language-tab 1))])
  )

; Dyck language tab
(define dyck-language-tab
  (new vertical-panel% [parent tab-panel]))

(new message% [parent dyck-language-tab]
       [label "Dyck language D:"])

(define dyck-grammar-message
  (new message% [parent dyck-language-tab]
       [label ""]
       [auto-resize #t]))

(new message% [parent dyck-language-tab]
       [label "Rules:"])

(define dyck-rules-message
  (new message% [parent dyck-language-tab]
       [label ""]
       [auto-resize #t]))
```

```
(new button% [parent dyck-language-tab]
     [label "Next >"]
     [callbackλ
      ( (button event)
        (switch-tab tab-panel cf-language-tab 2))]
     )

; Context-free language tab
(define cf-language-tab
  (new horizontal-panel% [parent tab-panel]))

(define cf-rule-input
  (new vertical-panel% [parent cf-language-tab]))

(new message% [parent cf-rule-input]
     [label "Context-free grammar rules:"])

(define cf-rule-list
  (new vertical-panel% [parent cf-rule-input]))

(for ([i 3])
    (cf-add-rule-input cf-rule-list))

(new button% [parent cf-rule-input]
     [label "Add rule"]
     [callback λ( (button event)
                 (cf-add-rule-input cf-rule-list))])

(new button% [parent cf-rule-input]
     [label "Update rules"]
     [callback λ( (button event)
                 (cf-process cf-rule-list)
                 (send cf-next enable #t))])

(define cf-description
  (new vertical-panel% [parent cf-language-tab]))

(new message% [parent cf-description]
     [label "Context-free language L':"])

(new message% [parent cf-description]
     [label ""])


(new message% [parent cf-description]
     [label "Rules:"])

(new message% [parent cf-description]
     [label ""])
```

```
(define cf-next
  (new button% [parent cf-description]
       [label "Next >"]
       [enabled #f]
       [callbackλ
        ( (button event)
          (switch-tab tab-panel r-language-tab 3))])
  )

; Regular language tab
(define r-language-tab
  (new vertical-panel% [parent tab-panel]))

(new message% [parent r-language-tab]
     [label "Regular language L0:"])

(new message% [parent r-language-tab]
     [label ""])

(new message% [parent r-language-tab]
     [label "Rules:"])

(new message% [parent r-language-tab]
     [label ""])

(new button% [parent r-language-tab]
     [label "Next >"]
     [callbackλ
      ( (button event)
        (switch-tab tab-panel run-tab 4))])

; Run tab
(define run-tab
  (new vertical-panel% [parent tab-panel]))

(define run-field
  (new text-field% [parent run-tab]
       [label "Input string"]))

(new button% [parent run-tab]
     [label "Run"]
     [callback λ( (button event) (run-automata))])

(define run-result
  (new message% [parent run-tab]
       [label "Waiting for run result"]
       [auto-resize #t]))

(define run-result-panel
  (new horizontal-panel% [parent run-tab]
       [alignment '(center center)]))
```

```
; hide all irrelevant tabs
(send dyck-language-tab show #f)
(send cf-language-tab show #f)
(send r-language-tab show #f)
(send run-tab show #f)

;start
(send main-frame show #t)
```

ДОДАТОК Б

Ілюстративний матеріал

Рисунок Б.1 – Слайд 1

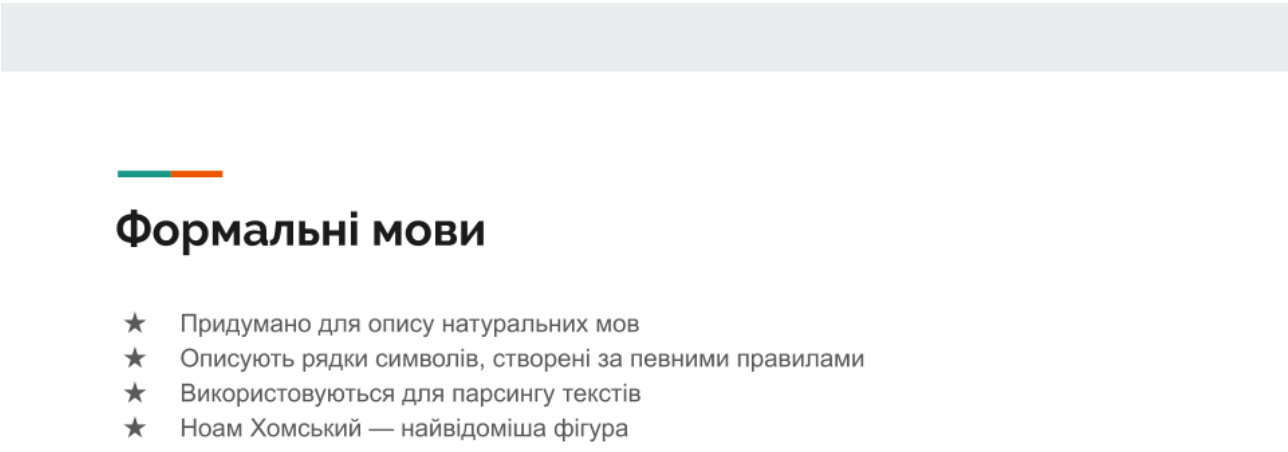

Рисунок Б.2 – Слайд 2

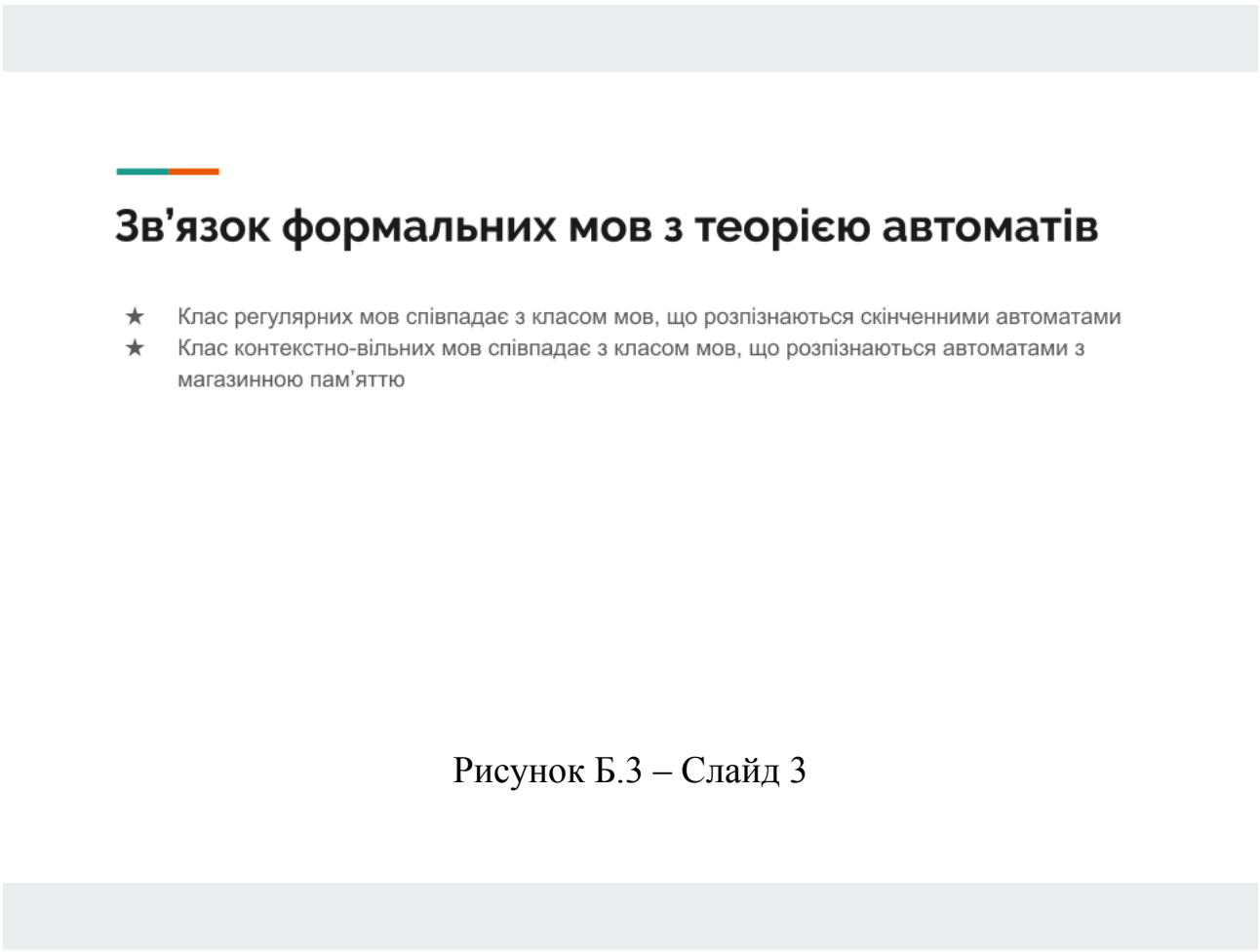

Рисунок Б.3 – Слайд 3

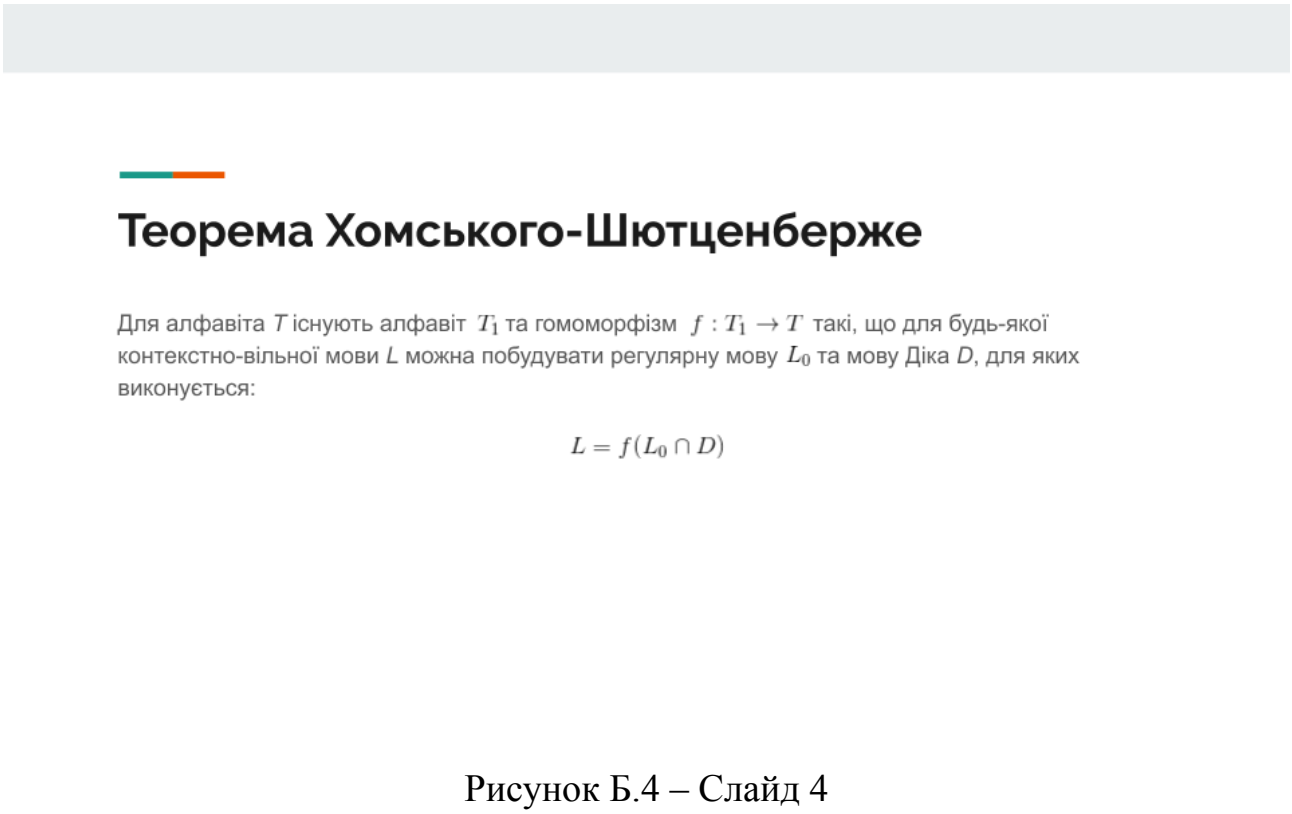

Рисунок Б.4 – Слайд 4

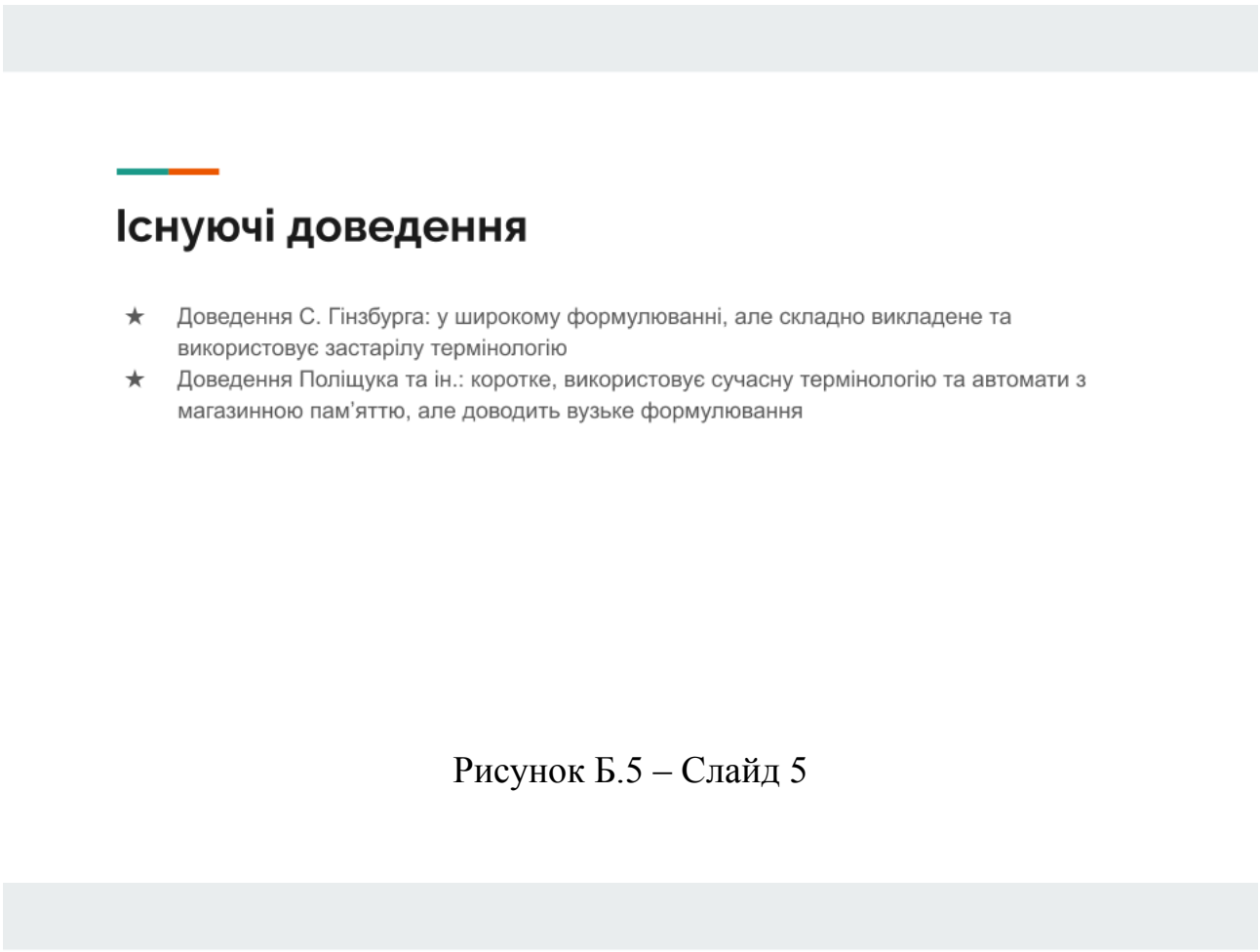

Рисунок Б.5 – Слайд 5

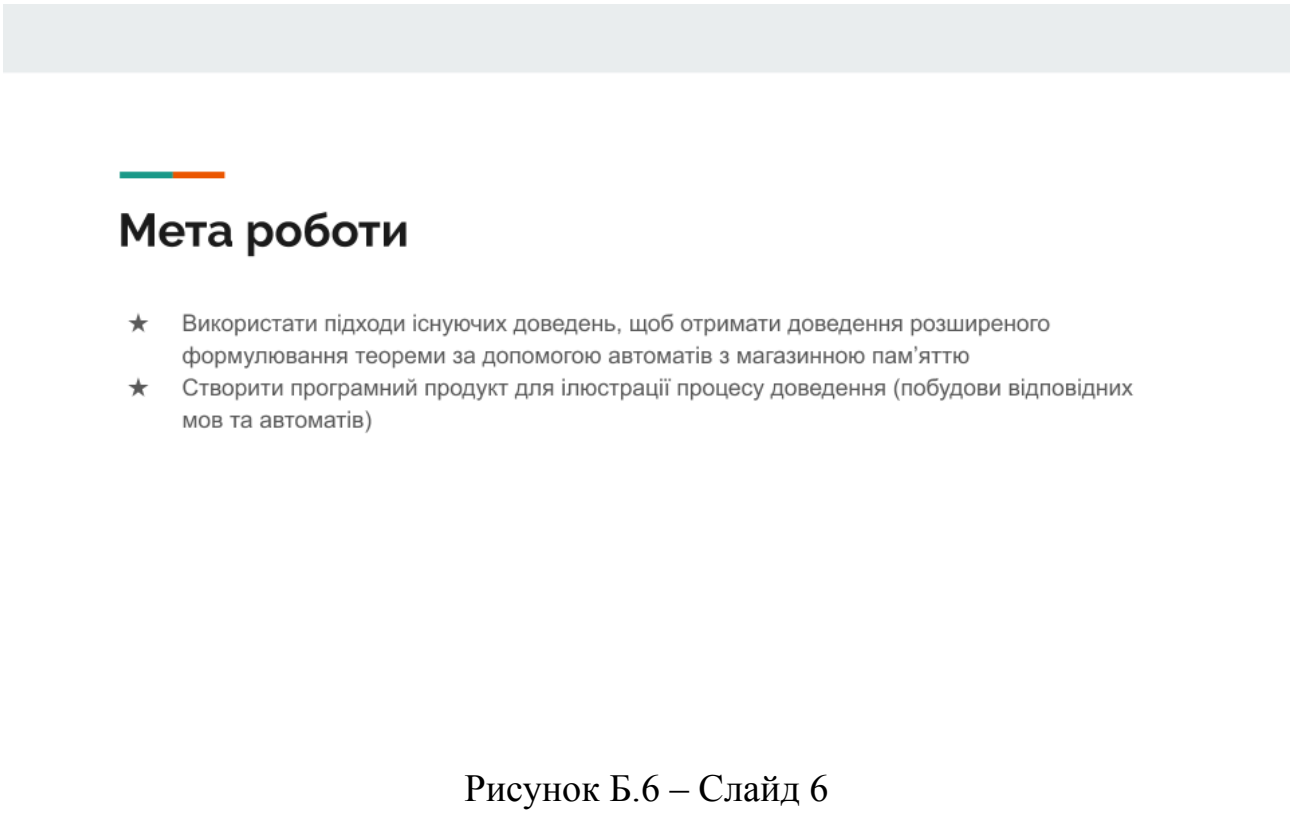

Рисунок Б.6 – Слайд 6

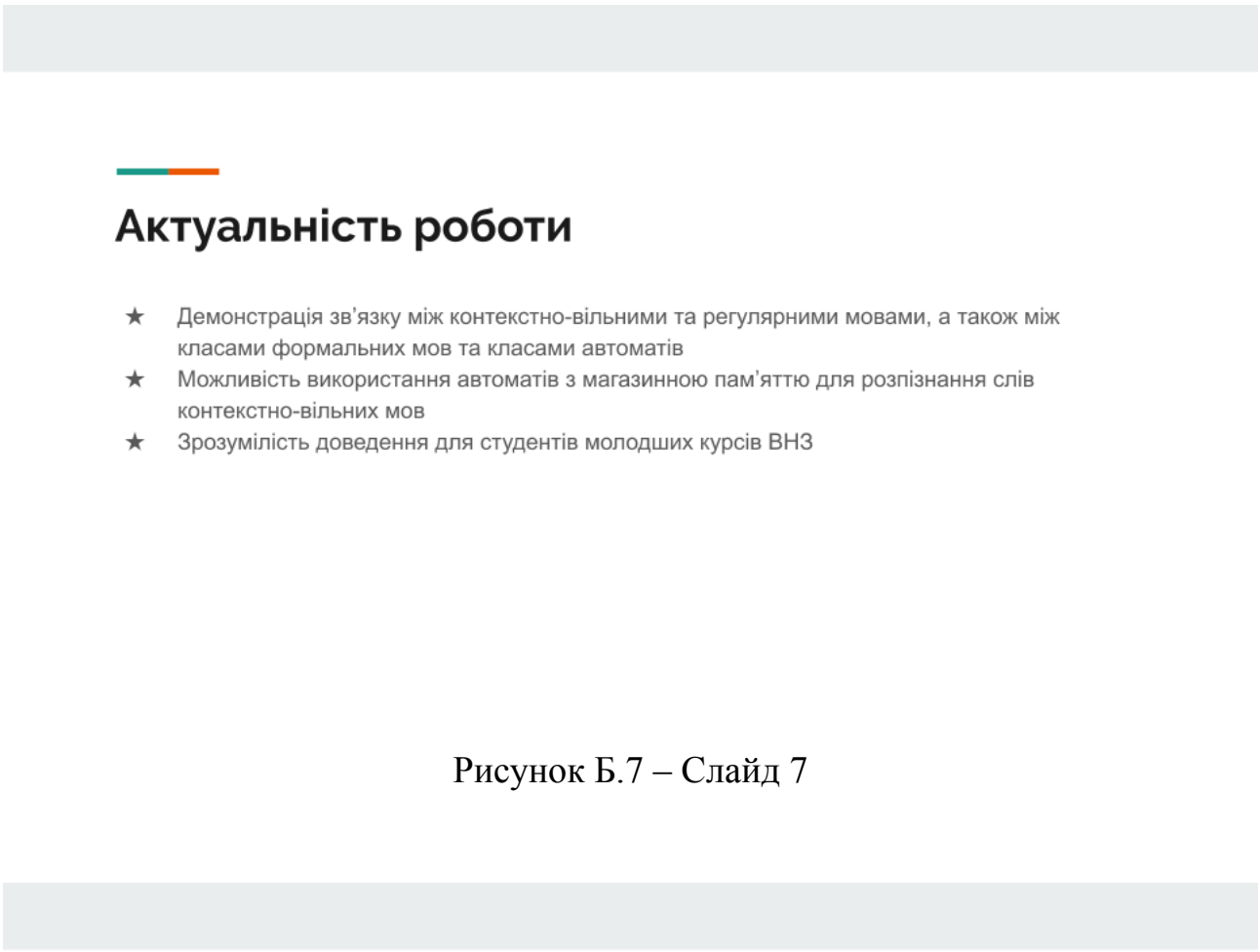

Рисунок Б.7 – Слайд 7

Рисунок Б.8 – Слайд 8

Порівняння класів мов з класами автоматів

- ★ Регулярні мови: розгортаються зліва направо; завжди один нетермінальний символ; можна передати станом скінченного автомата
- ★ Контекстно-вільні мови: можна розгортати зліва направо; може бути декілька нетермінальних символів; можна передати стеком автомата

Рисунок Б.9 – Слайд 9

Зв'язок між мовою Діка та стеком

- ★ Правильну дужкову послідовність можна передати послідовністю взаємодій зі стеком
- ★ Кожній парі дужок *a*, *a'* можна співставити стековий символ *A*, що додаватиметься, коли дужку відкрито, і вилучатиметься, коли її закрито
- ★ Таким чином, логічно використовувати мову Діка для опису стека автомата, що відповідає контекстно-вільній мові

Рисунок Б.10 – Слайд 10

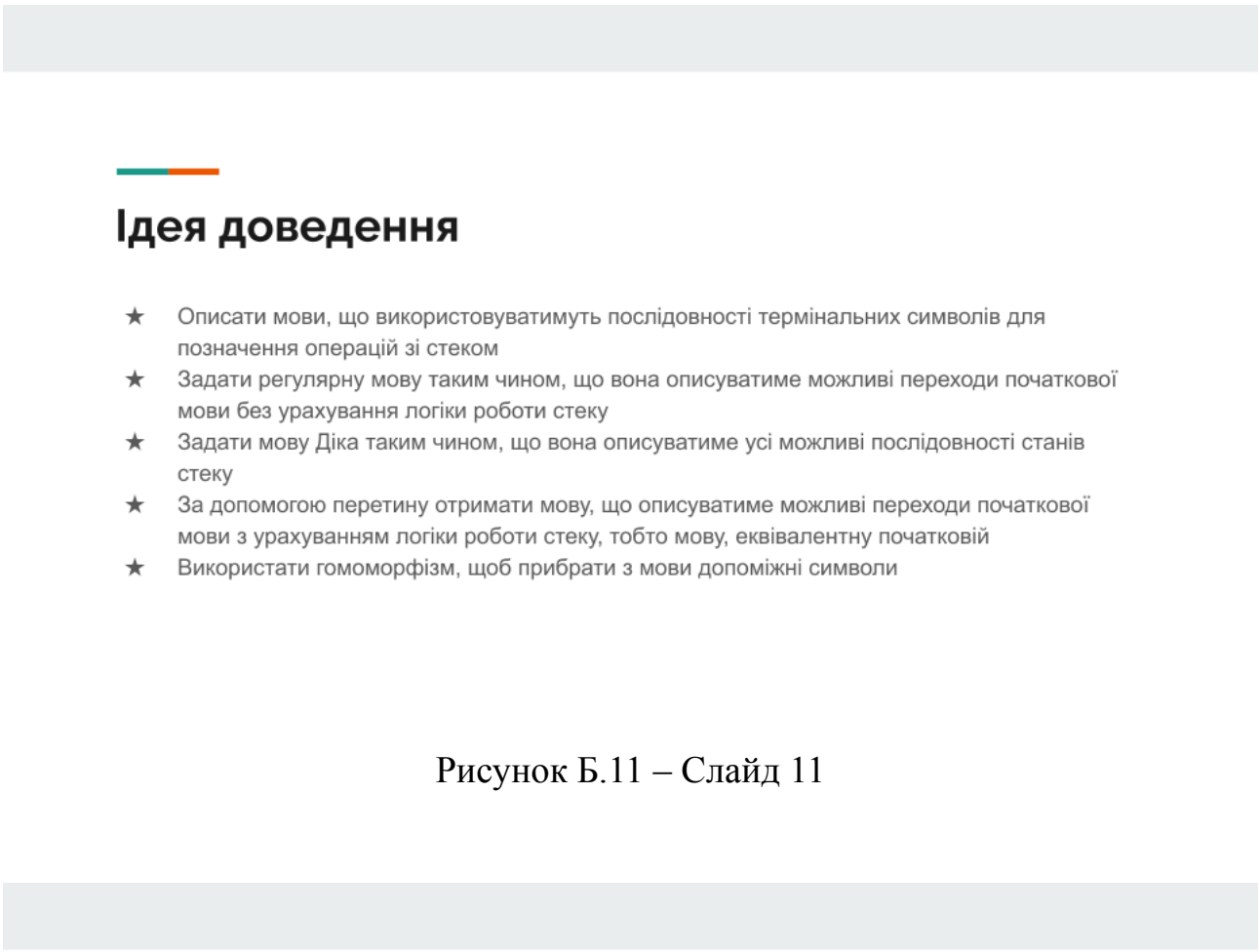

Рисунок Б.11 – Слайд 11

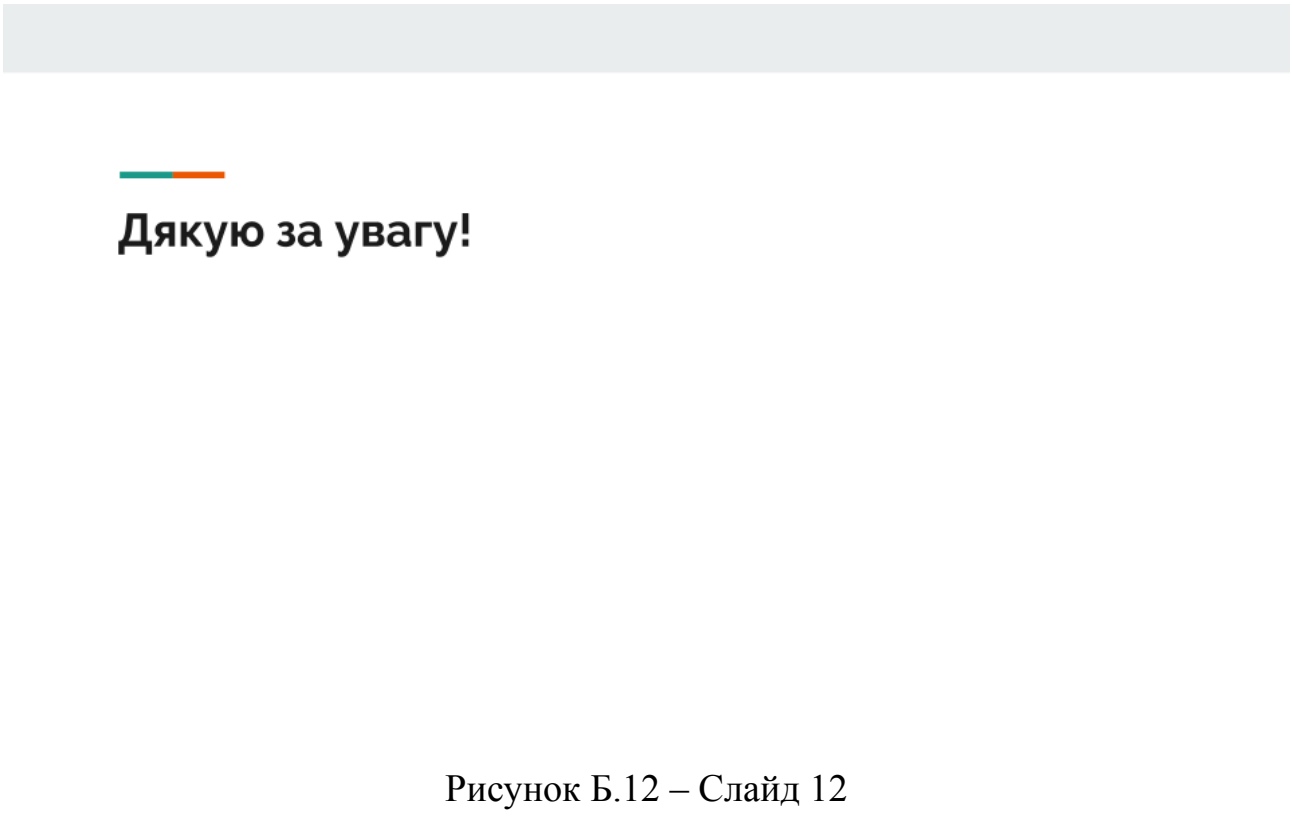

Рисунок Б.12 – Слайд 12